EXHIBIT 1

PROVISIONAL APPLICATION

FILED APRIL 14, 2001

"DATA ADAPTOR"

Inventors:

Robert Broderson, et al.

EXHIBIT 1

UNITED STATES PROVISIONAL PATENT APPLICATION

FOR

DATA ADAPTER

Inventors:
Robert Broderson
Mark Coyle
Sanjin Tulac

Prepared by:
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Seventh Floor
Los Angeles, California 90025-1026
(408) 720-8598

Attorney's Docket No: 5306P030Z

"Express Mail" mailing label number: EL627471279 US
Date of Deposit: April 14, 2001
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Assistant Commissioner for Patents, Washington, D. C. 20231

Glenn E. Von Tersch
(Typed or printed name of person mailing paper or fee)

[illegible]
(Signature of person mailing paper or fee)

April 14, 2001
(Date signed)

DATA ADAPTER

[01] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

Field of the Invention

[02] The invention primarily relates to the field of data processing and more particularly to transforming and updating data represented in a relational database based on data represented in a hierarchical form.

Description of the Related Art

[03] While it is well-known to represent data in the form of a relational database, maintaining data in such a database can be a challenge. Outside sources of data can provide new or updated information, but this can be of limited utility when the only option for updating a database is to have a person manually view the outside data source and then update the database. As such, a method for adapting data for update or insertion in a database may be useful.

BRIEF DESCRIPTION OF THE DRAWINGS

[04] The present invention is illustrated by way of example and not limitation in the accompanying figures.

[05] Figure 1 illustrates an embodiment of a potential dataflow allowing for upsert or synchonization of a relational database from an outside source.

[06] Figure 2A illustrates an embodiment of data flow from a hierarchical structure to a relational database structure.

[07] Figure 2B illustrates an alternate embodiment of data flow from a hierarchical structure to a relational database structure.

[08] Figure 3A illustrates an embodiment of a definition of an integration object.

[09] Figure 3B illustrates an embodiment of an instance of an integration object according to the definition of Figure 3A.

[10] Figure 4A provides a flow diagram of an embodiment of a method of upserting or synchronizing an object.

[11] Figure 4B illustrates an embodiment of an integration component and corresponding userkeys.

[12] Figure 4C illustrates an embodiment of an integration object and related userkeys.

[13] Figure 4D illustrates an embodiment of a process of extracting a userkey for use with an integration component.

[14] Figure 5 provides a flow diagram of an embodiment of a method of upserting or synchronizing children associated with an object.

[15] Figure 6 illustrates an embodiment of a set of actions associated with a set of conditions.

[16] Figure 7A provides a flow diagram of an embodiment of a method of generating integration objects from a relational database.

[17] Figure 7B provides a flow diagram of an embodiment of a method of generating child components in a integration object.

DETAILED DESCRIPTION

[18] A data adapter is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention can be practiced without these specific details. In other instances, structures and devices are shown in block diagram form in order to avoid obscuring the invention.

[19] Reference in the specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase “in one embodiment” in various places in the specification are not necessarily all referring to the same embodiment, nor are separate or alternative embodiments mutually exclusive of other embodiments. Moreover, various features are described which may be exhibited by some embodiments and not by others. Similarly, various requirements are described which may be requirements for some embodiments but not other embodiments.

[20] A data adapter may be used to translate data between a relational database and a hierarchical data structure. Furthermore, the data adapter may be used to update or synchronize the relational database and the hierarchical data structure, thus allowing for coordination of data sources which may have radically different internal structures.

[21] Figure 1 illustrates an embodiment of a potential dataflow allowing for update of a relational database from an outside source. External data 110 may be a

data source such as another relational database, an XML document, a stored user file such as a spreadsheet or word processing document, or other data. Converter 120 represents a converter which may be utilized to transform the data of external data 110 into data represented as integration objects, such as integration objects 130. Integration objects 130 are a set of objects having a hierarchical interrelationship, with each object embodying data and/or methods of accessing data of the object or of accessing related objects. Data adapter 140 represents a converter suitable for upserting or synchronizing data from an integration object into a database or transforming database entries (records) into integration objects. RDBMS 150 represents a relational database system, which stores a variety of data in records having various relationships.

[22] Note that the converter 120 may be implemented in a variety of ways, some of which are discussed in the application entitled "Integration Objects" which was filed on the same day as the filing of this application and which is subject to assignment to the same assignee to which this application is subject to assignment. For purposes of this discussion, it is assumed that the integration objects may be created in a straightforward manner, and may be useful as a representation of data stored in a relational database. Furthermore, note that the terms upsert and synchronize appear frequently. Upserting is a combination of updating existing data or inserting new data in a data destination, without deleting data from the data destination which may not exist in the data source. Sychronization is a combination of taking data from a data source, updating existing data or inserting new data relative to data in a data destination, and deleting data from the data destination which is not present in the data source.

[23] In one embodiment, the integration objects 130 are formed as a tree of objects, each having pointers to related objects and embodying both data and methods of accessing the data and related objects. In such an embodiment, an object may be referred to as a root object, indicating that it does not depend on any other object. Such an object typically represents an entry associated with a key or record which is viewed as an original or top-level record. Other objects may be referred to as children of a root, and such child objects will be pointed to by either a root object or by another child object. Typically, a parent of a child object is an object which has a pointer to the child object. Thus, a root object may be expected to be a parent to one or more child objects, and a child object may also be a parent to one or more child objects. With respect to correlations between a root object or child object and database structure, such correlations may or may not be straightforward. Given the relations within a relational database, a first record may be chosen as corresponding to a root object, resulting in a second record appearing to be a child object, while an alternate choice of the second record as the root object may result in the first record appearing to be a child object.

[24] Figure 2A illustrates an embodiment of data flow from a hierarchical structure to a relational database structure. Integration objects 230 are acceptable as an input to data adapter 240, which then upserts or synchronizes the data from the integration objects 230 into the database 250. Alternately, data from the database 250 is used as an input to the data adapter 240, which then produces integration objects 230 as a representation of the data from database 250. Note that the data from database 250 may be a subset of the entire contents of database 250, such as a set of

entries selected by a user or requested by a controlling or requesting program for example. Furthermore, the structure of the integration object 230 is defined in metadata available from the metadata repository 245. This metadata may be used by the data adapter to determine not only the structure of the integration object 230, but also how the data embodied in the integration object 230 maps to data in the relational database 250.

[25] Figure 2B illustrates an alternate embodiment of data flow from a hierarchical structure to a relational database structure. Data adapter 270 is illustrated as including a component level adapter 272, field level adapter 274, object manager 276 and data manager 278. In general, object managers and data managers are relatively well-known types of software modules which may be used to access objects or data in a relational database, and which generate SQL (Sequel) queries for use with the relational database. In one embodiment, the component level adapter 272 and field level adapter 274 each are used to process data at various levels of the hierarchical representation of the integration objects 230 or the relational representation of the database 250. The component level adapter 272 may, in one embodiment, process components or objects, while the field level adapter 274 may process individual fields or portions of components or objects.

[26] The metadata repository 280 may be made available to any or all portions of the data adapter 270. The metadata corresponding to the integration object 230 contained in the metadata repository 280 provides an indication of the structure of the integration object 230 and an indication of the relationship between the structure of the integration object 230 and the relationships among data and tables in the relational

database. Thus, the data manager 278 and object manager 276 may utilize the metadata to determine how queries should be structured and how resulting data should be parsed. Similarly, the component 272 and field level adapters 274 may utilize the metadata to determine which fields or components to access, both where data should come from and where data should go. Alternatively, the metadata may be viewed as controlling the various portions of the data adapter 270 by determining how the data flows between the integration object 230 and the relational database 250.

[27] Inherent in the ability to process data is some understanding of the representation of that data prior to processing. For the data adapter to utilize data in integration objects or represent data as integration objects, a format for such integration objects must be provided, and it will be appreciated that this format may be varied according to a variety of design choices. In some embodiments, this format is defined in metadata held in a repository and available to portions of the data adapter.

[28] Figure 3A illustrates an embodiment of a definition of an integration object. Within the integration object are components and fields, of which fields are not illustrated in Figure 3A. An account 310 may have zero or more contacts 320 and zero or more business addresses 350. Each contact 320 may have zero or more activities 330 and zero or more personal addresses 340. As will be appreciated, definitions for integration objects may be flexible, allowing for various formats and hierarchical relationships. In this illustration, contact 320 is a child of account 310, which is its parent. Moreover, contact 320 is parent to each of children activities 330 and personal address 340. It will be appreciated that each of the children in Figure 3A will be provided as components associated with the parent of the child in question. Moreover,

it will be appreciated that hierarchical data formats cover a broad spectrum, each of which may be utilized in a fairly straightforward manner to store data and allow for retrieval of stored data. Within the integration object, each component has associated with it a userkey as will be described in more detail below.

[29] Figure 3B illustrates an embodiment of an instance of an integration object according to the definition of Figure 3A. An instance of an integration object may be viewed as an object which has a structure conforming to the definition of the integration object. Account 310A has a series of three children, each of which is a contact 320A and two other children, each of which is a business address 350A. The first contact 320A has a series of three children, each of which is an activity 330A and another series of two children, each of which is a personal address 340A. Similarly, the second contact 320A has a series of three children, each of which is an activity 330B. Moreover, the third contact 320A has a single child which is a personal address 340C. While each contact 320A has an identical structure to other contacts 320A, the data embodied therein may be unique or otherwise distinct. Similarly, the activities 330A and 330B may be expected to have a common structure, but embody varying data. Note that in the exemplary illustration associated userkeys are not shown.

[30] Figure 4A provides a flow diagram of an embodiment of a method of upserting or synchronizing an object. At block 410, a userkey is extracted from the object, as described further below. At block 420, the userkey is looked up or found in the database, providing for one of two proper responses, namely that a record corresponds to the userkey (and thus may need to be modified) or that no corresponding record exists. Thus, the userkey allows for access to data (record(s))

which already exist in the database and correspond to the object in question. At block 430, a determination is made as to whether the corresponding record was found. If not, at block 440, a new record is inserted, having a userkey as extracted and having any other data which is embodied in the object in question. Note that this may involve creating or updating related entries for children of the object in a manner similar to that described with respect to Figure 5. If a corresponding record is found, at block 450, the record is updated to reflect the data in the object, and examination of children of the object may result in additional updates to related records or entries in the database, again in a manner similar to that described with respect to Figure 5. This process of either inserting a new record or updating an existing record may be referred to as an upsert process or operation.

[31] Figure 4B illustrates an embodiment of an integration object definition and related userkey definitions. Integration object 1050 includes a set of integration components 1060 (components). Each integration component 1060 includes a set of integration component fields 1080 (fields). Associated with each integration component 1060 is a set of one or more userkeys 1070, the structure of which may be derived from the components. Associated with each integration component field 1080 is a set of userkey fields 1090, the structure of which may be derived from the integration component fields 1080. Each userkey 1070 may be made up of one or more userkey fields 1090.

[32] With respect to Figure 4C, an embodiment of an integration component definition and corresponding userkeys is illustrated. Account 1010 is the root component of an object. This Account component contains a set of fields 1030, and a

set of userkeys 1020. Each userkey 1020 consists of a set of userkey fields 1040, and each userkey field is associated directly with a single field 1030. The first userkey 1020 labeled "A" consists of a single field called 'Integration Id', and association is provided by a single userkey field 1040. The second userkey 1020 labeled "B" is associated with userkey fields 1040, which point to fields 1030 labeled "Name" and "Location" (labels not shown). As will be appreciated, userkey fields 1040 allow for assembly of userkeys in various combinations in conjunction with fields 1030 and potential combinations thereof.

[33] Userkey definitions are an integral part of an integration component definition, which is in turn an integral part of an integration object definition. Which userkey is applicable to a particular integration component is implied by the structure of the data embodied in that component, as defined by the userkey extraction algorithm described later.

[34] The userkeys associated with an integration object may each be used for purposes of matching data within the database to the integration component instances. As long as a userkey can be used to form a query which may be submitted to the database and return a related record from the database, the userkey is potentially useful for purposes of upserting or synchronizing data. However, some userkeys which are combinations of some fields will not be useful in forming a query, because that data does not exist in the database. Thus, there is a need for an opportunity to use multiple userkeys to find the data in question. A single userkey for an integration component may be defaulted to including a few specified fields based on design of the object, and the user of the system may change this design based on performance. However, rather

than require the user to constantly intervene when a single userkey is not producing the desired results (access to data), multiple userkeys are used.

[35] The queries thus formed may take advantage of both inner and outer joins allowed in SQL queries. Fields in the object may correspond to fields in the tables of a relational database in various ways. Thus, a key within the database for a field in the object may include a key related to a first table (for part of the field in the object) and a key related to a second table (for another part of the field in the object). It may be expected that a given component or integration object maps to a first table, but that it also maps to a variety of other tables due to the differences in structure between the database and the integration object. The key related to the second table or any other table is a foreign key relative to the first table, and thus inner and outer joins may be used to resolve the foreign keys and allow access to the corresponding data in the database.

[36] Figure 4D illustrates an embodiment of a process of extracting a userkey for use with an integration component. At block 910, a list of userkeys is created for the component of the integration object which is subject to search in the database, and the list is based on the definition of the integration object. As a performance optimization, this list can be cached in memory, and thus block 910 needs to be performed only once per integration component definition. At block 920, the first userkey in the list of userkeys is made the current userkey. At block 930, an attempt to find each field of the userkey in the integration component instance is made, thus validating that the userkey relates directly to this instance of the integration object. If the search succeeds, the userkey is used in conjunction with the corresponding values of the fields in question at

block 970. If the search at block 930 fails, a check is made at block 940 as to whether this was the last userkey in the list. If not, the next userkey in the list is made the current userkey at block 960, and the search of block 930 is repeated. Finally, if this is the last userkey in the list, at block 950 an error is generated indicating that a valid userkey cannot be found, and that the integration object can not be upserted or synchronized with the database.

[37] Figure 5 provides a flow diagram of an embodiment of a method of upserting or synchronizing children associated with an object. At block 510, child records associated with an entry or record are looked up in a database, resulting in a set or list of child records. At block 515, the first child record of the list of child records is made the current child record. At block 520, the current child record is matched to a corresponding component in the hierarchy of components. Note that this matching may be done based on a definition of such a hierarchy and matching with a corresponding component as defined, or may be done in a different manner, such as searching through a hierarchy for a matching component. At block 525, if no match is found, the child record is left undisturbed, or is deleted, depending on whether the type of update is intended to be destructive (synchronize) or merely additive (upsert). In the case of a deletion, this may result in some sort of cascaded delete if appropriate, as the child record in question may have data dependencies within the relational database which must be either preserved or deleted as a whole. At block 530, if a match is found, that match is detected, and at block 535, the child record is updated to reflect the data in the corresponding component.

[38] At block 540, a determination is made as to whether unprocessed child records remain. If such unprocessed child records remain, the next unprocessed child record is then regarded as the current child record, and the process returns to block 520. If no unprocessed child records remain, unmatched components in the object or hierarchy of objects are found. If such unmatched components remain, those components are inserted into the database as child records corresponding to the components in question and having data from the components in question. If no unmatched components remain, or all unmatched components have been inserted, the process ends at block 580.

[39] Note that in some embodiments, the processing of child records or components will typically have a recursive nature, due in part to the hierarchical structure of the object. Thus, children at one level may be processed one-by-one, with children of children processed before the next child at a given level is processed (thus traversing the tree while processing). Alternatively, the children at the next level may be processed after all children at the current level are processed (in a depth first manner for example).

[40] In one embodiment, child records are grouped according to the type of the child record, and each child record of a first type is processed before any child record of a second type is processed. However, such processing need not affect whether the processing of a given child record has a recursive nature, such that the processing of child records of a first child occurs before processing of a second child record (and corresponding child records of the second child record).

[41] Figure 6 illustrates an embodiment of a set of actions associated with a set of conditions. In the table, X represents presence in a data representation, a dash represents absence. As will be apparent, if a component or field or other item of data is present in both the input objects and the existing RDBMS (database), the data in the database is updated to reflect the data in the object. If the data exists only in the object representation form, the data is inserted in the database. If the data exists only in the database, one of two things occurs. One, the data may be deleted if the process is to conform the database to the objects (synchronize). Two, the data may be left undisturbed if the process is only to update data to reflect any data actually present in the object (upsert).

[42] When data is to be extracted from the database and embodied in an integration object, one embodiment of the basic method is to first find the root entry and create an object for that. Next, the types of children available are found, and iteration on each child record of a given type is performed to create the child records, and then the next type of child record is processed. However, the children may be processed in a recursive manner, such that the children of a child record of a first type will be created prior to processing of a child record of a second type. Figures 7A and 7B illustrate an alternate embodiment of this method.

[43] With respect to Figure 7A, the overall process is illustrated. At block 710, records satisfying the specification for the root integration component are found. If no records are found, the process stops at block 790 and no integration object or component is created. If one or more records are found, at block 720, the first root record (record corresponding to the specification for the root integration component) is

made the current root record. An integration object based on the current root record is created at block 730.

[44] At block 740, direct children records of the current root record are found. If no such children records exist, at block 770 a determination is made as to whether root records are left for processing. If no root records remain for processing, at block 790 the process stops. If another root record remains for processing, at block 780 the next root record is made the current root record, and the process moves back to block 730. Note that the query for direct children records is made by the component level adapter, which goes through the object manager and data manager to find the direct children records in one embodiment. However, the metadata associated with the integration object may block the query for children, thus indicating that no direct children records are found due to the structure of the object and not due to the presence or absence of children records within the database.

[45] If direct children records are found, then at block 745 the first child record is made the current child record. The child records are ordered based on the type of child record, such that all child records of a first type are processed before any child records of a second type are processed. At block 750, the current child record is processed, which may include the process illustrated in Figure 7B in one embodiment. Such processing may include creating a component of the integration object related to the child record and/or creating any necessary fields of a component of the integration object related to the child record or children of the child record.

[46] At block 760, a determination is made as to whether any direct children have not been made the current direct child record. If a direct children record has not

been made the current direct child record, at block 765 the next direct child record is made the current direct child record, and processing of the current direct child record occurs at block 750. If no direct children records have not been made the current direct child record, the process goes to block 770 to check whether root records still remain to be processed as described previously.

[47] In one embodiment, processing of children records may be accomplished as illustrated in Figure 7B. At block 810, a component is created based on a parent record, such as the child record to be processed at block 750 of Figure 7A. Creating the component includes creating fields of the component, and filling in data. This may be accomplished by the component level adapter and the field level adapter working in tandem. Whether the data from the database becomes a component or field is dependent on the metadata definition of the integration object, the component and field level adapters match the data to the structure defined.

[48] At block 820, direct child records of the parent record are found. If no direct child records are found, the process stops at block 870. If direct child records are found, at block 830 the first direct child record is made the current record. Note that the ordering of the child records is based on the type of child record, as was described previously with respect to Figure 7A.

[49] At block 840, the current record is processed, which, in one embodiment, primarily involves a recursive call to the same routine implementing Figure 7B, with the current record regarded as the parent record at block 810. Thus, processing includes finding children of the current record and creating a component and/or fields related to the current record. Alternatively, the processing may involve creating the component

and/or fields associated with the current child record and recursively finding and processing children of the current child record.

[50] At block 850, a determination is made as to whether any direct child records have not been made the current child record. If so, the next direct child record is made the current child record at block 860 and the process goes to block 840 for processing of the current record. If no direct child records have not been made the current child record, the process terminates at block 870.

[51] Note that reference has been made to finding records within the database at various points in this description. Typically, records are found in the database using SQL queries to the database. For example, finding children of a first record may be accomplished through a SQL query specifying that all records having a parent id corresponding to the first record. The metadata associated with the integration object provides an indication as to which tables within the database should be queried based on both the definition of the object and the position of the corresponding component in the hierarchy. Furthermore, the inner and outer joins mentioned previously may be used as part of these queries.

[52] Attached hereto as Exhibits A-D are the following:

Exhibit A - Design Specification for Hierarchy Support in EAI (17 pages);

Exhibit B - Design Specification for Enterprise Application Interfaces (42 pages);

Exhibit C - Using Integration Objects (46 pages);

Exhibit D - Using the EAI Siebel Adapter (9 pages);

[53] These exhibits (A, B, C, and D) are incorporated herein by reference.

[54] In the foregoing, the present invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the present invention. In particular, the separate blocks of the various block diagrams represent functional blocks of methods or apparatuses and are not necessarily indicative of physical or logical separations or of an order of operation inherent in the spirit and scope of the present invention. The present specification and figures are accordingly to be regarded as illustrative rather than restrictive.

CLAIMS

What is claimed is that which has been described in the foregoing and equivalents thereof.

Design Document for Hierarchy Support in EAI

Exhibit A

Business Requirements

It may be useful to support configuration, pricing and ordering of complex products. Complex products are a hierarchy of other products. Hierarchy in this context means 'has-a' relationship. For example, a car is a complex product that encompasses an engine, a chassis, wheels etc. The engine can be treated as a complex product itself, since it encompasses other products, like valves, pistons etc. Another well-known example would be a computer, consisting of a monitor, system unit, keyboard, mouse etc.

In order to support operations on complex products, it may be useful to have a generic mechanism to represent the hierarchy. Obviously, this mechanism can then be reused wherever hierarchy is useful, not only in complex products. It may be useful to support only homogeneous hierarchies, which means that all the elements in the hierarchy are instances of the same business component. There should be no limit on the number of levels in the hierarchy.

Implementation Overview

From the end user's perspective, hierarchy support enables the following operations:

- View, browse and edit complex products and other hierarchical data through a specialized applet
- Validation of complex products through the Product Configurator
- Calculation of prices for complex products through Product Pricer
- Sending and receiving orders for complex products to and from external back-office application through appropriate EAI Connector

User Procedures

Most of the user interaction for complex products will be done through the specialized hierarchy applet. Details of the GUI design are going to be explained elsewhere.

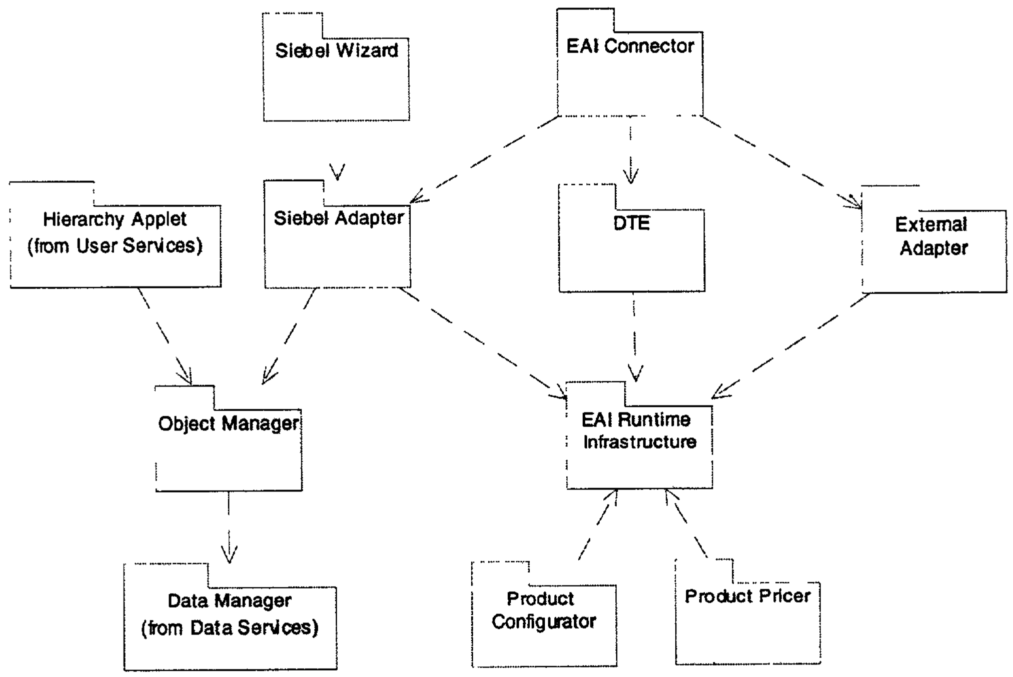
Generally, calls to operations will be done by pressing a button in the GUI (e.g, 'Validate', 'Calculate Prices', or 'Send the Order to SAP')

Administrator Procedures

No hierarchy-specific procedures for administrators are envisioned.

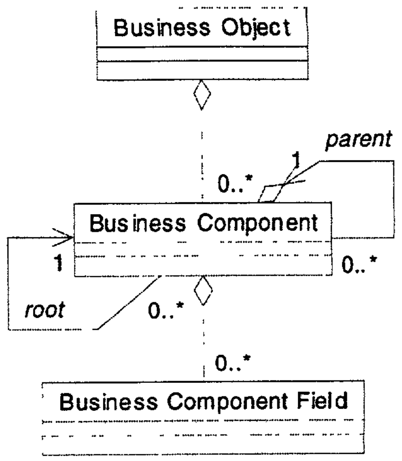
High Level Architecture

The diagram below describes the components involved in the hierarchy support, and their dependencies. It is easy to see that the central component is EAI Runtime Infrastructure, which will be used to store instances of hierarchies in memory, to facilitate their transport and modification between other components of the system. This has significant scalability advantages compared to having components communicating through the RDBMS.



Summary of Affected Components

Programs

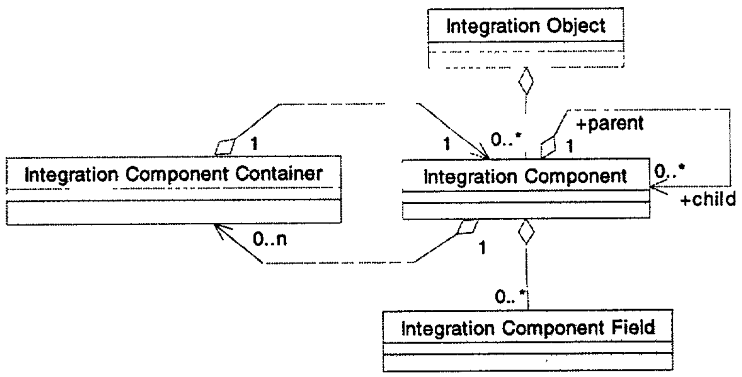


Object Manager (modified): preferably should accommodate the Business Object model as depicted on the right. For performance reasons, the pointer (mapped to a foreign key in the underlying table) to the root Business Component of the hierarchy is denormalized. This will speed up the queries on the whole hierarchy, since no traversal will be necessary to fetch all the Business Component instances that make up a complex product in a simple query.

EAI Runtime Infrastructure (modified): preferably should support representation, traversal, and modification of hierarchies as property sets and XML. XML natively supports hierarchies by nesting, e.g.:

```
<Product>
    <Name>Mercedes SLK 320</Name>
    <Product>
        <Name>V6 Engine 3250 ccm</Name>
    </Product>
</Product>
```

There are two points worth noting in this example:



- We don't use <ListOfProducts> container element inside of the hierarchy. It's not needed, since the whole hierarchy is nested inside of the root element, so an empty hierarchy would just not include any nested elements of the root element type. Having container element inside of the hierarchy would just make parsing more difficult.
- Having explicitly denormalized pointers to the root and parent element inside of each hierarchy element is both problematic (since pointers are not naturally supported by XML) and unnecessary (since XML elements are not being stored as separate records in a RDBMS)

The same arguments are applicable to the property set representation. This leads to the Integration Object model that EAI Runtime Infrastructure needs to support, as depicted on the right diagram.

Note that child Integration Components don't have any pointers to either their parents, or the root of the hierarchy. Integration Component Container is used to represent the parent-child relationships (analogous to links in Business Objects), while the hierarchy is represented directly by the tree of Integration Components.

Siebel Adapter (modified): preferably should be able to convert data between Integration Object model and Business Object model. This means that semantics of its operations should be augmented:

- *Upsert/Synchronize*: the hierarchy should be processed from the root towards the leaves, and the ROW_ID column values of all processed hierarchy nodes need to be cached, to enable efficient setting of the root and immediate parent foreign key fields. Two modes of operations should be supported:

    - *Full:* the whole hierarchy is specified in the input Integration Object, and the corresponding Business Object is modified accordingly. This enables changing the structure of the hierarchy, as well as the contents of the Business Components that make up the hierarchy.

    - *Partial:* only Integration Components that correspond to the Business Components in the hierarchy that need to be synchronized are specified. This implies that no structural changes to the hierarchy are done, and Business Components that need to be changed are looked up using their user key fields. This is meant as a performance optimization that avoids huge Integration Objects for small (e.g. incremental snapshot) changes. If needed, this feature can probably be deferred for later implementation.

- *Delete*: deleting a node in the hierarchy tree implies deleting the whole sub-tree whose root is the node being deleted. Whether records in the underlying BusComp are actually cascade deleted, or just updated depends on the definition of the user key: if the user key includes the foreign key from the child to the parent, then cascade delete is appropriate. Otherwise, records should be updated.

- *Query:* only root Business Component can be specified to get the whole hierarchy.

Siebel Wizard(modified): preferably should create hierarchy related meta data in Integration Objects based on the meta data in Business Components.

DTE (Data Transformation Engine) (modified): preferably should keep the hierarchy data while doing the transformation. Depending on the external system, hierarchy to non-hierarchy and vice versa transformation may be useful.

EAI Meta Data Design

Hierarchy support at design time should be added to the definitions of:

- Business Component
- Integration Object

At the Business Component level, field that points to the parent's *Id* field should be marked as such. Currently, that is accomplished by adding a user property *Hierarchy Parent Field* to the Business Component, whose value is the name of the field.

At the Integration Object level, the meta-data needed by Siebel adapter are incorporated under *Int Component Key* type. Optionally, a new attribute: *Key Type* may be added, and two new key types may be provided to support hierarchy meta-data:

- Hierarchy Parent Key
- Hierarchy Root Key

For each entry of these two types, there should be a single entry in the subordinated *Int Component Key Field* type. Its *Field Name* attribute points to the actual *Integration Component Field* that holds the key value.

Alternative way of specifying that a component can contain hierarchy is to set user property *IsHierarchyComponent* on the Integration Component to *Y*. This mechanism is useful for the cases when keys do not make sense, such as XML Integration Objects created by the DTD Wizard.

Worth noting is that a "hierarchy-enabled" Integration Component can also be a parent of other Integration Components. This means that "homogeneous" and "heterogeneous" hierarchies can be intermixed, and that every node in the homogeneous hierarchy can have heterogeneous children. This is actually an essential requirement for support of Complex Products, since extended attributes of a product are stored in a separate Business/Integration Component than the product itself (to support 1:M relationship). So, the hierarchy of products is homogeneous, but every product can also have heterogeneous children. In case of XML Integration Objects any arbitrary element can have itself as a child and other elements as children as well. Such elements are mapped to *Homogeneous Hierarchy* components.

Siebel Wizard Algorithm

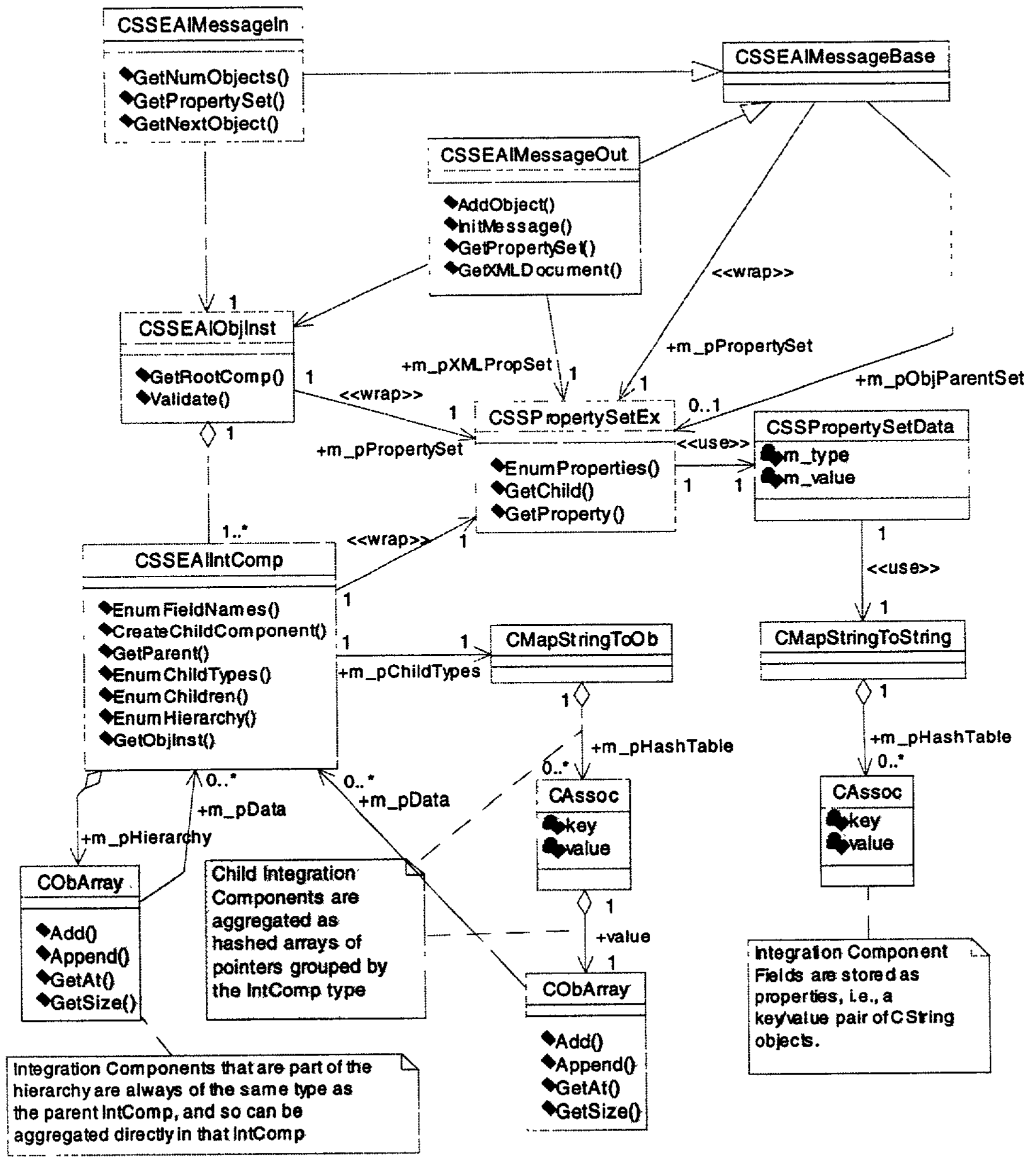
To support hierarchy, Siebel Wizard has to create an Integration Component Key of type *Hierarchy Parent Key* for each Integration Component which maps to a Business Component with the user property *Hierarchy Parent Field.* The Integration Component Key Field needs to point to the Integration Component Field that maps to the field specified by the value of this user property.

The Integration Component Field that is referenced by the *Hierarchy Parent Key* needs to be created as Siebel system field, so that it doesn't get into the outbound Integration Object Instance (since it's a Siebel implementation detail).

EAI Runtime Infrastructure Design

EAI Runtime Classes

The following diagram depicts the most important classes that carry out the responsibilities of EAI Runtime Infrastructure. Please, note that only part of the design that is significant for the hierarchy support has been shown. The main change is addition of a `CObArray` instance directly into `CSSEAICompInst` class, pointed to by a member variable `m_pHierarchy`. Also, as described more detailed in the next section, new method `EnumHierarchy` should be added to support traversing the hierarchy implemented through `m_pHierarchy`.



EAI Runtime API

It seems that the actual EAI Runtime API should be only slightly augmented in the class `CSSEAICompInst`. In addition to methods `EnumChildTypes`, `EnumChildren`, and `ChildCount`, used to gain access to the Integration Component instances aggregated in the current Integration Component instance *(aggregated IntComps)*, new public methods need to be added for creating, traversing, and changing the Integration Components in the hierarchy *(hierarchy IntComps)*.

Current (v6.x) interface for creating and traversing aggregated IntComps that looks like this:

```
// Factory method to create a child component, given the name of its type.
virtual ErrCode CreateChildComponent(const CString& compTypeName,
                                     CSSEAICompInst*& pCompOut) = 0;
// Factory method to create a child component type. This is done automatically
// by CreateChildComponent(), but must be called explicitly to create a
// component type entry without any component instances.
virtual ErrCode CreateChildCompType(const CString& compTypeName) = 0;

// Enumerate over the names of the child types present in this component,
// in the order the component definitions in the integration component
// metadata.
// Returns FALSE if no type was returned (because there are none left).
// If an error occcurred, errorOut is set to the appropriate error code.
BOOL EnumChildTypes(BOOL bGetFirst, CString& typeNameOut, int& enumPos,
                    ErrCode& errorOut) const;
// Enumerate over the names of the child types present in this component.
// Returns FALSE if no type was returned (because there are none left).
// If an error occcurred, errorOut is set to the appropriate error code.
BOOL EnumChildTypes(BOOL bGetFirst, CString& typeNameOut, POSITION& enumPos,
                    ErrCode& errorOut) const;
// Get the number of children of the specified type. bExists is true when
// the child type is present in the component.
ErrCode ChildCount(const CString& typeName, UINT& numChildren,
                   BOOL& bExists) const;
// Enumerate through the child components of the specified type.
// Returns FALSE if pCompOut is null (because there are no more children left).
BOOL EnumChildren(BOOL bGetFirst, const CString& typeName,
                  const CSSEAICompInst*& pCompOut, int& enumPos,
                  ErrCode& errorOut) const;
BOOL EnumChildren(BOOL bGetFirst, const CString& typeName,
                  CSSEAICompInst*& pCompOut, int& enumPos,
                  ErrCode& errorOut);
```

In some embodiments, this interface should be expanded with the following:

```
// Factory method to create a child component in the hierarchy
virtual ErrCode CreateChildInHierarchy(CSSEAICompInst*& pCompOut) = 0;

// Enumerate through the hierarchy of IntComps whose immediate parent is this IntComp.
// Returns FALSE if pCompOut is null (because there are no more immediate children).
// The order in which children components have been created is preserved.
// This method is const, since it will not modify the wrapped property set, but it
// actually may change the state of the wrapper instance by creating the container
// it's enumerating on, if it hadn't been already created!
BOOL EnumHierarchy(BOOL bGetFirst, const CSSEAICompInst*& pCompOut, int& enumPos)const;
BOOL EnumHierarchy(BOOL bGetFirst,       CSSEAICompInst*& pCompOut, int& enumPos);
```

In addition, the semantics of the following methods should be changed, while keeping their current signatures:

```
// Copy all the field values found in the source component. If bRecurse == TRUE,
// also copy all the aggregated children, as well as the whole hierarchy contained
// by the source component
// Note: source component wrapper may actually be modified in the process!
ErrCode Copy(const CSSEAICompInst* pSourceComp, BOOL bRecurse = TRUE);

// Get the number of children of the specified type. bExists is true when
// the child type is present in the component.
ErrCode ChildCount(const CString& typeName, UINT& numChildren, BOOL& bExists) const;
```

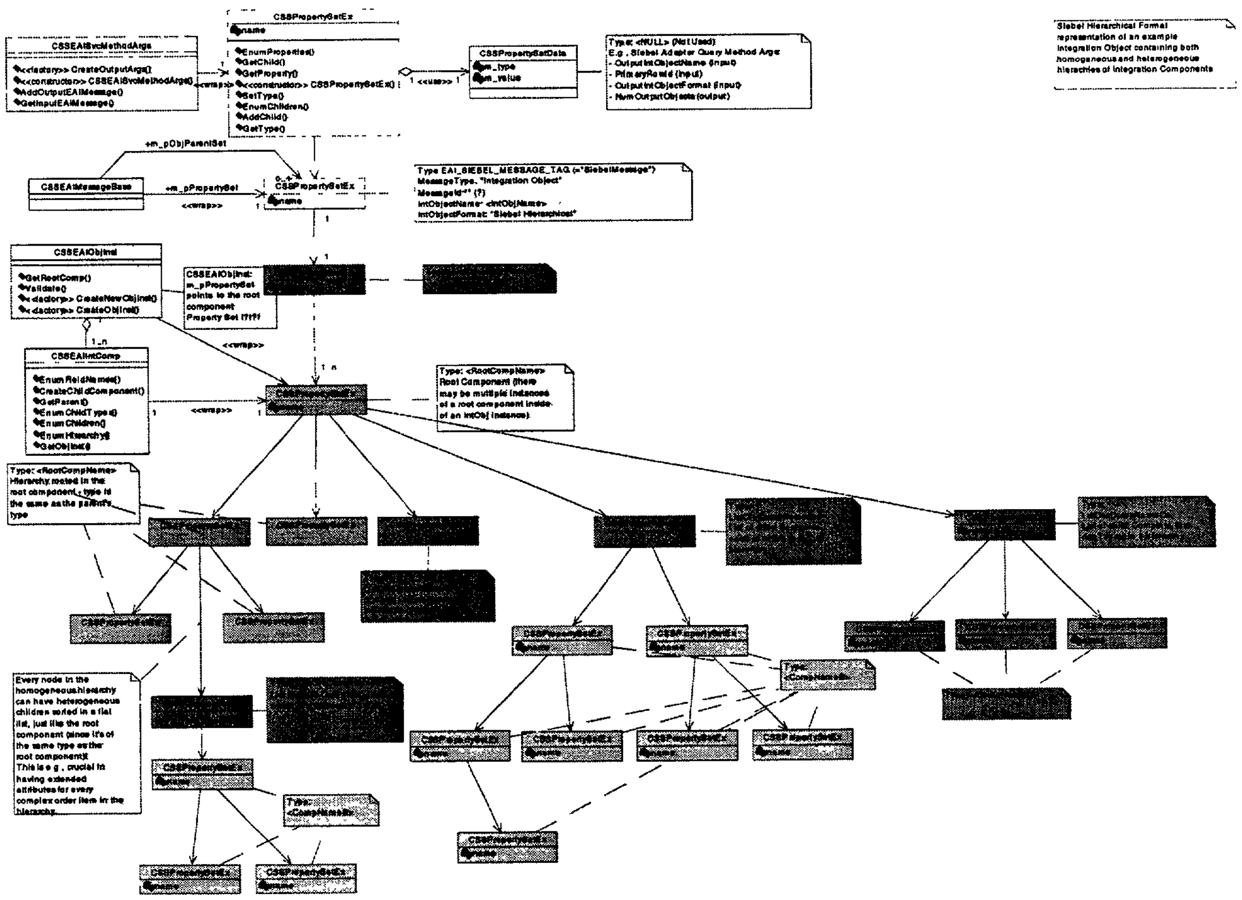
This minimal addition to `CSSEAICompInst` should be enough to manipulate the hierarchy, complemented with these methods already existing in v6.x:

```
// Enumerate over the names of the fields present in this component.
// Returns FALSE if no field was returned (because there are none left).
virtual BOOL EnumFieldNames(BOOL bGetFirst, CString& fieldNameOut,
                            POSITION& enumPos) const = 0;
// Get the value of the specified field. If the field has no value, an
// empty string is returned. If the field name does not match a legal field
// of the component, an error is returned. If the field name is valid for the
// component, but it is not present in the component, bPresent will be set
// to false.
virtual ErrCode GetFieldValue(const CString& fieldName, CString& fieldValueOut,
                              BOOL& bPresent) const = 0;
// Set the value of the specified field.
virtual ErrCode SetFieldValue(const CString& fieldName,
                              const CString& fieldValueStr) = 0;
```

EAI Runtime Implementation

Creating the hierarchy of Integration Component instances (MessageOut) is easy, since the array of wrapper classes is created at the same time as the related Property Sets. The tricky part of the implementation is parsing the MessageIn, which is passed to the Business Service as a plain hierarchy of Property Sets. Example of such a hierarchy of Property Sets in the "Siebel Hierarchical Format" is depicted on the diagram below.

Existing design of EAI runtime infrastructure creates wrappers one level at a time. After the wrapper for the root


CSSEAISvcMethodArgs
<<factory>> CreateOutputArgs()
<<constructor>> CSSEAISvcMethodArgs()
AddOutputEAIMessage()
GetInputEAIMessage()
CSSPropertySetEx
name
EnumProperties()
GetChild()
GetProperty()
<<constructor>> CSSPropertySetEx()
SetType()
EnumChildren()
AddChild()
GetType()
<<wrap>>
<<use>>
CSSPropertySetData
m_type
m_value
Type: <NULL> (Not Used)
E.g. Siebel Adapter Query Method Args:
- OutputIntObjectName (input)
- PrimaryRowId (input)
- OutputIntObjectFormat (input)
- NumOutputObjects (output)
Siebel Hierarchical Format representation of an example Integration Object containing both homogeneous and heterogeneous hierachies of Integration Components
+m_pObjParentSet
CSSEAIMessageBase
+m_pPropertySet
<<wrap>>
0..+
CSSPropertySetEx
name
Type EAI_SIEBEL_MESSAGE_TAG (="SiebelMessage")
MessageType. "Integration Object"
MessageId="" (?)
IntObjectName= <intObjName>
IntObjectFormat: "Siebel Hierarchical"
CSSEAIObjInst
GetRootComp()
Validate()
<<factory>> CreateNewObjInst()
<<factory>> CreateObjInst()
CSSEAIObjInst: m_pPropertySet points to the root component Property Set !?!?!
1..n
<<wrap>>
CSSEAIIntComp
EnumFieldNames()
CreateChildComponent()
GetParent()
EnumChildTypes()
EnumChildren()
EnumHierarchy()
GetObjInst()
Type: <RootCompName> Root Component (there may be multiple instances of a root component inside of an IntObj instance)
Type: <RootCompName> Hierarchy rooted in the root component - type is the same as the parent's type
Every node in the homogeneous hierarchy can have heterogeneous children sorted in a flat list, just like the root component (since it's of the same type as the root component)! This is e.g. crucial in having extended attributes for every complex order item in the hierarchy.
Type: <CompName>
CSSPropertySetEx
name

component has been created, the wrappers for all (heterogeneous) children are created all at once by the implementation of the virtual method `CSSEAICompInst::CreateChildWrapperComps()`. This method creates the hash table of arrays pointing to child wrappers and initializes member variable `CSSEAICompInst::m_pChildTypes` with a pointer to it. This member variable is thus used to indicate the state of the component instance – whether the wrappers for child components have been created or not.

The same approach has been applied for the creation of wrappers for component instances that make up the hierarchy – in this case the member variable is called `CSSEAICompInst:: m_pHierarchy`, and `CSSEAICompInst::CreateHierarchyWrapperComps()` is the method that initializes it.

Siebel Adapter Design

Query

Hierarchy support in the query method of Siebel Adapter means the ability to get the whole hierarchy as a single Integration Object. To do that, Siebel Adapter should:

1. based on Integration Component definition, understand that it can contain a hierarchy
2. recursively query the whole hierarchy in the Business Component and create the corresponding hierarchy Integration Component instances
3. query the heterogeneous children *for each* node of the homogenous hierarchy in the BusComp

Step 1 can be implemented by looking up the Int Component Key of type *Hierarchy Parent Key*. In Siebel Adapter Integration Objects, these keys should always consist of a single Int Component Key Field.

BusComp API for hierarchy support (available since build 10519) should be used to support hierarchy in Siebel Adapter as well. It detects the hierarchy in current workset, and hides all records that are not at the top level. Siebel Adapter can then iterate through top level records, and call 'Expand' method recursively to get the records of next level. One problem is that if the search spec is excluding any records in the hierarchy, BusComp API will just ignore them. To work around this, Siebel Adapter needs to use *Root Id* to expand the search specification it would normally create. So, in the presence of a hierarchy with *Hierarchy Root Key* defined, it needs to execute more than one query: the first will find all rows that satisfy the query as specified by the input integration object. Then, for each record retrieved, a new query should be executed to find the whole tree that this record is part of[1].

Synchronize

Supporting hierarchy in the Synchronize method of Siebel Adapter means the ability to accept the whole hierarchy as a single Integration Object and recreate that hierarchy in the Siebel Database. To do that, Siebel Adapter should:

1. based on Integration Component definition, understand that it can contain a hierarchy, and figure out which keys to use as Parent Id and Root Id
2. recursively query the hierarchy in the Integration Object, compare it with the hierarchy in the Business Component, and insert/update/delete records in the BusComp as needed
3. query and synchronize the heterogeneous children *for each* Integration Component of the homogenous hierarchy

[1] Worth noting is that the record found doesn't even need to be at the root level. If the only root level records are to be searched, then parent Id or root Id field should be specified as null in the input search specification.

Step 2 needs to cache the values for Parent Id and Root Id in memory, since they are not explicitly contained in the Integration Object instance.

Unit Tests

1. Create Integration Object using Siebel Wizard

Test:

1. Create and lock a new project called “Homogenous Hierarchy Unit Test”
2. Start the Use the Siebel Wizard to create Integration Object “Complex Quote” based on the Business Object “Quote”. Select the following Integration Components:
    - `Quote`
        - `Quote Item`

3. Inactivate all fields but the following:
    - `Quote:`
        - `Name`
    - `Quote Item:`
        - `Integration Id`
        - `Quantity Requested`
        - `Parent Quote Item Id`

4. Inactivate all user keys and all their key fields except the following:
    - `Quote:`
        - `User Key:1`
            - `Name`
    - `Quote Item:`
        - `User Key:1`
            - `Integration Id`

Expected Result:

Siebel Wizard creates an Integration Component as usual, but for the Integration Component “Quote Item” it also creates an *Int Component Key* entry named “Parent Key” of type “Hierarchy Parent Key”. This entry should contain a single *Int Component Key Field* entry that points to the field “Parent Quote Item Id”. No other Integration Components should have any keys of this type defined.

2. Insert a Single Hierarchy using Siebel Adapter

Test:

1. Compile .srf that the *Siebel Client* will use to include the Integration Object
2. Run the Siebel Client with the newly compiled .srf file
3. Run in Siebel Adapter method ‘**Synchronize**’ in the Business Service Simulator with a following input property set:

```
<?Siebel-Property-Set EscapeNames="true" ?>
<PropertySet>
      <SiebelMessage
       MessageId="1"
       MessageType="Integration Object"
```

```
         IntObjectName="Complex Quote"
         IntObjectFormat="Siebel Hierarchical">
              <ListOfComplex_spcQuote>
                   <Quote Name="Test Quote 1">
                        <ListOfQuote_spcItem>
                             <Quote_spcItem
                              Integration_spcId="1"
                              Quantity_spcRequested="1">
                                   <Quote_spcItem
                                    Integration_spcId="1.1"
                                    Quantity_spcRequested="2">
                                   </Quote_spcItem>
                             </Quote_spcItem>
                        </ListOfQuote_spcItem>
                   </Quote>
              </ListOfComplex_spcQuote>
     </SiebelMessage>
</PropertySet>
```

Expected Result:

Run ODBCSQL with the following query:

```
SELECT Q.NAME, QI.INTEGRATION_ID, QI.QTY_REQ, QIP.INTEGRATION_ID PAR_INT_ID
FROM S_DOC_QUOTE Q
LEFT JOIN S_QUOTE_ITEM QI ON Q.ROW_ID = QI.SD_ID
LEFT JOIN S_QUOTE_ITEM QIP ON QIP.ROW_ID = QI.PAR_SQ_ITEM_ID
WHERE Q.NAME = 'Test Quote 1'
ORDER BY INTEGRATION_ID;
```

The result should look like this:

```
NAME                                                INTEGRATION_ID         QTY_REQ                PAR_INT_ID
--------------------------------------------------- ---------------------- ---------------------- ----------------------
Test Quote 1                                        1                      1.0000000              NULL
Test Quote 1                                        1.1                    2.0000000              1
--------------------------------------------------- ---------------------- ---------------------- ----------------------
```

The result can also be verified using the Quotes screen, Line Items view.

3. Update the Single Hierarchy using Siebel Adapter

Test:

1. Run **Synchronize** method of Siebel Adapter in the Business Service Simulator with the following input property set:

```
<?Siebel-Property-Set EscapeNames="true" ?>
<PropertySet>
      <SiebelMessage
       MessageId="2"
       MessageType="Integration Object"
       IntObjectName="Complex Quote"
       IntObjectFormat="Siebel Hierarchical">
            <ListOfComplex_spcQuote>
                  <Quote Name="Test Quote 1">
                        <ListOfQuote_spcItem>
                              <Quote_spcItem
                               Integration_spcId="1">
                                    <Quote_spcItem
                                     Integration_spcId="1.1"
                                     Quantity_spcRequested="3">
                                          <Quote_spcItem
                                           Integration_spcId="1.1.1"
```

```
                                             Quantity_spcRequested="4">
                                            </Quote_spcItem>
                                      </Quote_spcItem>
                                      <Quote_spcItem
                                       Integration_spcId="1.2"
                                       Quantity_spcRequested="5">
                                      </Quote_spcItem>
                                </Quote_spcItem>
                          </ListOfQuote_spcItem>
                    </Quote>
              </ListOfComplex_spcQuote>
        </SiebelMessage>
</PropertySet>
```

Expected Result:

Run ODBCSQL with the following query:

```
SELECT Q.NAME, QI.INTEGRATION_ID, QI.QTY_REQ, QIP.INTEGRATION_ID PAR_INT_ID
FROM S_DOC_QUOTE Q
LEFT JOIN S_QUOTE_ITEM QI ON Q.ROW_ID = QI.SD_ID
LEFT JOIN S_QUOTE_ITEM QIP ON QIP.ROW_ID = QI.PAR_SQ_ITEM_ID
WHERE Q.NAME = 'Test Quote 1'
ORDER BY INTEGRATION_ID;
```

The result should look like this:

```
NAME                                               INTEGRATION_ID         QTY_REQ                PAR_INT_ID
-------------------------------------------------- ---------------------- ---------------------- ----------------------
Test Quote 1                                       1                      1.0000000              NULL
Test Quote 1                                       1.1                    3.0000000              1
Test Quote 1                                       1.1.1                  4.0000000              1.1
Test Quote 1                                       1.2                    5.0000000              1
-------------------------------------------------- ---------------------- ---------------------- ----------------------
```

4. Insert Additional Homogenous Hierarchies using Siebel Adapter

Test:

1. Run **Synchronize** method of Siebel Adapter in the Business Service Simulator with the following input property set:

```
<?Siebel-Property-Set EscapeNames="true" ?>
<PropertySet>
      <SiebelMessage
       MessageId="3"
       MessageType="Integration Object"
       IntObjectName="Complex Quote"
       IntObjectFormat="Siebel Hierarchical">
            <ListOfComplex_spcQuote>
                  <Quote Name="Test Quote 1">
                        <ListOfQuote_spcItem>
                              <Quote_spcItem
                               Integration_spcId="1"
                               Quantity_spcRequested="6">
                                    <Quote_spcItem
                                     Integration_spcId="1.1"
                                     Quantity_spcRequested="7">
                                          <Quote_spcItem
                                           Integration_spcId="1.1.1"
                                           Quantity_spcRequested="8">
                                          </Quote_spcItem>
```

```
                                                </Quote_spcItem>
                                                <Quote_spcItem
                                                 Integration_spcId="1.2"
                                                 Quantity_spcRequested="9">
                                                </Quote_spcItem>
                                          </Quote_spcItem>
                                          <Quote_spcItem
                                           Integration_spcId="2"
                                           Quantity_spcRequested="10">
                                                <Quote_spcItem
                                                 Integration_spcId="2.1"
                                                 Quantity_spcRequested="11">
                                                </Quote_spcItem>
                                          </Quote_spcItem>
                                          <Quote_spcItem
                                           Integration_spcId="3"
                                           Quantity_spcRequested="12">
                                          </Quote_spcItem>
                                    </ListOfQuote_spcItem>
                              </Quote>
                        </ListOfComplex_spcQuote>
                  </SiebelMessage>
</PropertySet>
```

Expected Result:

Run ODBCSQL with the following query:

```
      SELECT Q.NAME, QI.INTEGRATION_ID, QI.QTY_REQ, QIP.INTEGRATION_ID PAR_INT_ID
      FROM S_DOC_QUOTE Q
      LEFT JOIN S_QUOTE_ITEM QI ON Q.ROW_ID = QI.SD_ID
      LEFT JOIN S_QUOTE_ITEM QIP ON QIP.ROW_ID = QI.PAR_SQ_ITEM_ID
      WHERE Q.NAME = 'Test Quote 1'
      ORDER BY INTEGRATION_ID;
```

The result should look like this:

```
NAME                                              INTEGRATION_ID        QTY_REQ               PAR_INT_ID
------------------------------------------------- --------------------- --------------------- ---------------------
Test Quote 1                                      1                     6.0000000             NULL
Test Quote 1                                      1.1                   7.0000000             1
Test Quote 1                                      1.1.1                 8.0000000             1.1
Test Quote 1                                      1.2                   9.0000000             1
Test Quote 1                                      2                     10.0000000            NULL
Test Quote 1                                      2.1                   11.0000000            2
Test Quote 1                                      3                     12.0000000            NULL
------------------------------------------------- --------------------- --------------------- ---------------------
```

5. Query Multiple Homogenous Hierarchies using Siebel Adapter

Test:

Run **Query** method of Siebel Adapter in the Business Service Simulator with the following input property set:

```
<?Siebel-Property-Set EscapeNames="true" ?>
<PropertySet>
      <SiebelMessage
       MessageId="4"
       MessageType="Integration Object"
       IntObjectName="Complex Quote"
       IntObjectFormat="Siebel Hierarchical">
            <ListOfComplex_spcQuote>
                  <Quote Name="Test Quote 1">
                  </Quote>
            </ListOfComplex_spcQuote>
      </SiebelMessage>
</PropertySet>
```

Expected Result:

The output property set should look like this (possibly differently formatted):

```
<?Siebel-Property-Set EscapeNames="true" ?>
<PropertySet>
      <SiebelMessage
       MessageId="4"
       MessageType="Integration Object"
       IntObjectName="Complex Quote"
       IntObjectFormat="Siebel Hierarchical">
            <ListOfComplex_spcQuote>
                  <Quote Name="Test Quote 1">
                        <ListOfQuote_spcItem>
                              <Quote_spcItem
                               Integration_spcId="1"
                               Quantity_spcRequested="6">
                                    <Quote_spcItem
                                     Integration_spcId="1.1"
                                     Quantity_spcRequested="7">
                                          <Quote_spcItem
                                           Integration_spcId="1.1.1"
                                           Quantity_spcRequested="8">
                                          </Quote_spcItem>
                                    </Quote_spcItem>
                                    <Quote_spcItem
                                     Integration_spcId="1.2"
                                     Quantity_spcRequested="9">
                                    </Quote_spcItem>
                              </Quote_spcItem>
                              <Quote_spcItem
                               Integration_spcId="2"
                               Quantity_spcRequested="10">
                                    <Quote_spcItem
                                     Integration_spcId="2.1"
                                     Quantity_spcRequested="11">
                                    </Quote_spcItem>
                              </Quote_spcItem>
                              <Quote_spcItem
                               Integration_spcId="3"
                               Quantity_spcRequested="12">
                              </Quote_spcItem>
                        </ListOfQuote_spcItem>
                  </Quote>
            </ListOfComplex_spcQuote>
      </SiebelMessage>
</PropertySet>
```

6. Delete parts of the Hierarchies using Siebel Adapter

Test:

Run **Synchronize** method of Siebel Adapter in the Business Service Simulator with the following input property set:

```
<?Siebel-Property-Set EscapeNames="true" ?>
<PropertySet>
      <SiebelMessage
```

```
         MessageId="5"
         MessageType="Integration Object"
         IntObjectName="Complex Quote"
         IntObjectFormat="Siebel Hierarchical">
              <ListOfComplex_spcQuote>
                    <Quote Name="Test Quote 1">
                          <ListOfQuote_spcItem>
                                <Quote_spcItem
                                 Integration_spcId="1"
                                 Quantity_spcRequested="13">
                                      <Quote_spcItem
                                       Integration_spcId="1.2"
                                       Quantity_spcRequested="14">
                                      </Quote_spcItem>
                                <Quote_spcItem
                                 Integration_spcId="3"
                                 Quantity_spcRequested="15">
                                </Quote_spcItem>
                          </ListOfQuote_spcItem>
                    </Quote>
              </ListOfComplex_spcQuote>
        </SiebelMessage>
  </PropertySet>
```

Expected Result:

Run ODBCSQL with the following query:

```
SELECT Q.NAME, QI.INTEGRATION_ID, QI.QTY_REQ, QIP.INTEGRATION_ID PAR_INT_ID
FROM S_DOC_QUOTE Q
LEFT JOIN S_QUOTE_ITEM QI ON Q.ROW_ID = QI.SD_ID
LEFT JOIN S_QUOTE_ITEM QIP ON QIP.ROW_ID = QI.PAR_SQ_ITEM_ID
WHERE Q.NAME = 'Test Quote 1'
ORDER BY INTEGRATION_ID;
```

The result should look like this:

```
NAME                                          INTEGRATION_ID        QTY_REQ               PAR_INT_ID
--------------------------------------------- --------------------- --------------------- ---------------------
Test Quote 1                                  1                     13.0000000            NULL
Test Quote 1                                  1.2                   14.0000000            1
Test Quote 1                                  3                     15.0000000            NULL
--------------------------------------------- --------------------- --------------------- ---------------------
```

7. Delete all Hierarchies using Siebel Adapter

Test:

```
<?Siebel-Property-Set EscapeNames="true" ?>
<PropertySet>
      <SiebelMessage
       MessageId="6"
       MessageType="Integration Object"
       IntObjectName="Complex Quote"
       IntObjectFormat="Siebel Hierarchical">
            <ListOfComplex_spcQuote>
                  <Quote Name="Test Quote 1">
                        <ListOfQuote_spcItem>
                        </ListOfQuote_spcItem>
                  </Quote>
            </ListOfComplex_spcQuote>
      </SiebelMessage>
</PropertySet>
```

Expected Result:

Run ODBCSQL with the following query:

```
SELECT Q.NAME, QI.INTEGRATION_ID, QI.QTY_REQ, QIP.INTEGRATION_ID PAR_INT_ID
FROM S_DOC_QUOTE Q
LEFT JOIN S_QUOTE_ITEM QI ON Q.ROW_ID = QI.SD_ID
LEFT JOIN S_QUOTE_ITEM QIP ON QIP.ROW_ID = QI.PAR_SQ_ITEM_ID
WHERE Q.NAME = 'Test Quote 1'
ORDER BY INTEGRATION_ID;
```

The result should look like this:

```
NAME                                               INTEGRATION_ID         QTY_REQ                PAR_INT_ID
-------------------------------------------------- ---------------------- ---------------------- ----------------------
Test Quote 1                                       NULL                   NULL                   NULL
-------------------------------------------------- ---------------------- ---------------------- ----------------------
```

Design Specification for

Enterprise Application Interfaces

Exhibit B

1 Business Requirements

The goal of this project is to build a set of interfaces to SAP and Peoplesoft that allow Siebel users to synchronize customer, product, price, and employee data with the ERP system, to submit orders, and to track the status of orders. In the process of implementing this specific interface, the infrastructure will be developed to permit customers to customize/extend the ERP interfaces and to enable the integration of Siebel with other applications.

For detailed information on business requirements, please refer to the Marketing Requirement Documents *SAP R/3 System Interfaces* and *Peoplesoft Interfaces* by Chike Eleazu.

1.1 Goals

- Master data only (initial load and ongoing)
- ERP to Siebel only
- Object Snapshots
- No deletes (not provided by PeopleSoft)
- Full schema for DTE and Integration Objects.
- Entity mappings are read-only in tools. Implemented by DTE as SQL templates
- Field mappings are configurable by customers and up-gradable. Mappings are specified using an expression language that includes source fields, constants, truncation, concatenation, decode, and table lookup.
- Mappings are from PeopleSoft staging tables and IDOCs to EIM interface tables. These mappings will continue to work in the next release, but users may have to rewrite their maps if they want to map to BusComps or to some "Canonical Object".

2 High Level Architecture

The architecture that we have defined for Enterprise Integration includes the following types of components:

- *Siebel Interface Adapters* to get data in and out of Siebel. These adapters are built on top of existing interfaces (e.g. Interface Tables and Business Component Interfaces) and provide a layer of abstraction that handles the details specific to each interface.

- A *data transformation service* that can convert objects and relationships from one application to the objects and relationships of another application. These transformations can be categorized as either entity-level transformations or field-level transformations. An *component-level transformation* changes the relationships between objects in the source system, including combining objects, separating objects, or filtering objects. A *field-level transformation* is a function that operates on the fields of a single entity, combining them to produce an output field. For example, a join is an entity-level transformation, while a string concatenation is a field-level transformation.

  Application-specific or transport-specific control fields should not be visible to the data transformation service. These fields should be set by the appropriate adapter (e.g., the Siebel adapter, the format translator, or the transport adapter).

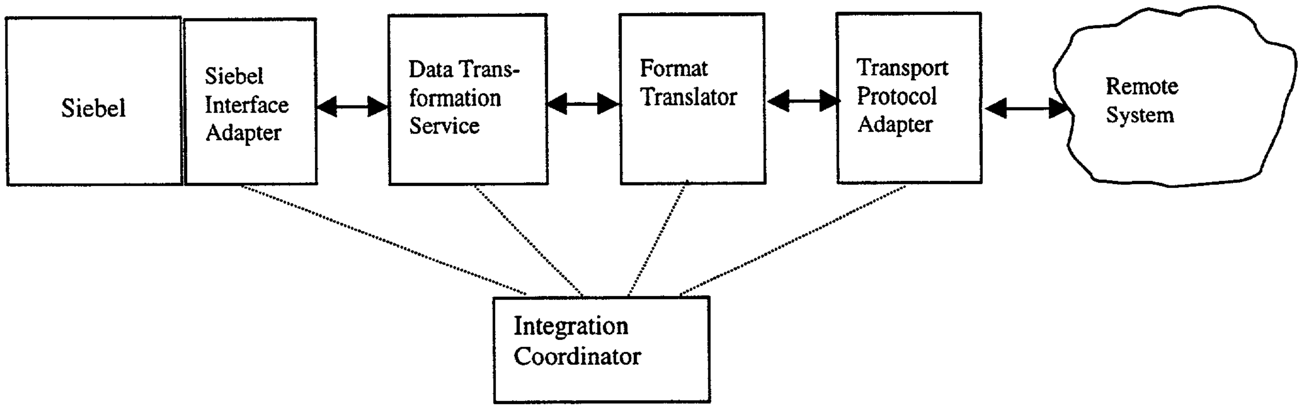
  The ideal data transformation service is capable of performing complex transformations, fast, metadata-driven, upgrade-able, and easy to use.

- A *format-translation service* converts objects between an internal, canonical representation and external representations. Ideally, this service should be independent from the data transformation service. As an example, let us suppose that a data transformation service accepts and generates objects using an in-memory tree structure. A format translation service would convert between this format and external formats such as XML, SAP IDOC, and application-specific formats.

- A *transport protocol adapter* performs the low-level communication with another application or interfaces to a standard API/protocol. Messages or objects represented in an external format are passed from the host application to this adapter or are accepted into the application from the adapter. Example transport protocols to which adapters could interface include HTTP, SAP ALE, MQ Series, and flat files.

- An *integration coordinator* guides data through the other components. Event handling / triggering, message routing, and task scheduling are provided by this component. For simple integrations, this component can be hard-coded, but larger integrations will require more sophisticated and flexible functionality.

For maximum flexibility, each of these components should be independent of the others. This simplifies the development of each component and allows components to reused in different interfaces. For example, an SAP IDOC format translator does not need to be aware of whether communication with the remote SAP system is occurring through SAP ALE, flat files, or some kind of middleware. The format translator can work with adapters for any of these protocols. Here is a block diagram of an idealized integration infrastructure:



For the Siebel 99.1 SAP and Peoplesoft integration projects, these idealized services will be provided by the following subsystems:

Component	Implementation for SAP Integration	Implementation for Peoplesoft integration
Siebel Interface Adapter	EIM Adapter	EIM Adapter
Data Transformation Service	DTE (Data Transformation Engine)	DTE
Format Translator	SAP adapter (IDOC parser)	Peoplesoft Adapter
Transport Protocol Adapter	SAP adapter (RFC/ALE libraries)	Peoplesoft Adapter
Integration Coordinator	EAI Server Component	EAI Server Component

2.1 Integration Objects

In an integration environment, object definitions may reside in Siebel's repository (tables and BusComps) or in external applications. Rather than forcing each adapter to know about each potential definition format, we will provide a single location within the Siebel repository to define objects exchanged between adapters. Objects defined using this sub-repository will be called *Integration Objects*. An integration object is composed of *components*, which are BusComp-like entities containing named and typed fields. The components of an integration object are related to one another by parent-child relationships.

Integration components and their fields can be linked to other (physical) objects in the repository. In Siebel 99.1, integration components can be linked to tables. In Siebel 2000, it will also be possible to link integration components to BusComps. The linking of integration components to physical objects provides a standard representation for Integration Adapters to use when exchanging objects. For example, if the components of an integration object are linked to table definitions, then an adapter can place its output data in those tables or other tables with the same definition. The adapter just needs to provide the consumer of the data with the integration object type and mappings from the integration components to the actual staging tables (if the tables referenced in the integration object definition were not used directly). In Siebel 2000, a standard in-memory representation of integration objects will also be provided.

Each integration component definition must include a unique key as well as a foreign key to its parent. These keys can include multiple fields and can overlap. De-normalized tables of the type used by EIM (where a table includes multiple components, but each row belongs to only one component) must have a normalized logical representation. The integration object repository is capable of representing a normalized object on top of these de-normalized tables.

2.2 Adapter Interfaces

We have defined a common interface for all adapter types (Siebel Interface adapters, data transformation adapters, format adapters, and protocol adapters.). This interface permits adapters to be implemented, tested and used independently, as recommended above. In the current model, adapters are unidirectional – request/data messages flow in one direction and response/status messages flow in the other. Each adapter has an input port and an output port. The routing of messages between adapters is established by connecting output ports to input ports at initialization time. After connections have been established, the adapters themselves can perform all message routing.

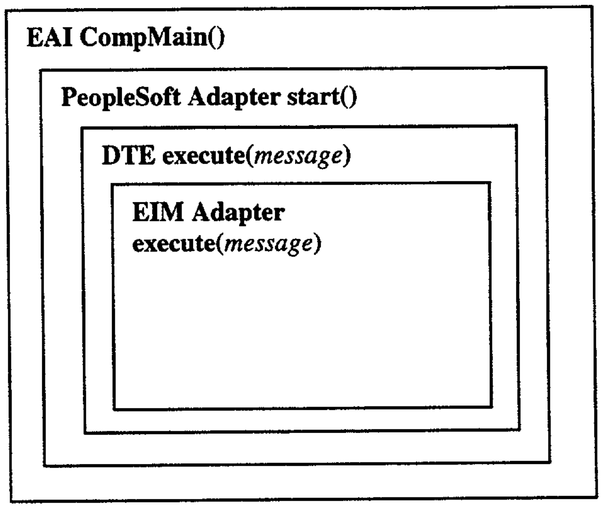
Each adapter must implement the following methods:

- *Query* – this method returns metadata describing the capabilities of the adapter.
- *Connect* – connects the input and output ports of the adapter to those of other adapters.
- *Initialize* – this method is called for each adapter after all the adapters have been instantiated and connected.
- *Start* – after all the adapters have been initialized, this method is called on the adapter acting as a message originator. This adapter should then retrieve messages and pass them onto the next adapter.
- *Execute* – this method is used to pass messages between adapters. The input of the call is a set of messages. The call returns status information and response messages (if any).

Adapter initialization is controlled by the EAI Server Component. When an instance of the EAI component is started, it performs the following operations:

1. It determines (from the repository) which adapters are needed for the requested integration process and instantiates them.
2. The EAI component then connects the adapters as required to route messages between Siebel interfaces and external applications.
3. The initialize() method is called on each adapter.
4. The originating adapter is started by calling its start() method. It can be started in one of two modes. In task mode, the adapter retrieves a batch of messages, passes them though the other adapters, and then exits. If there are no messages to be found in the initial check, an adapter running in task mode will exit with the status "Nothing to do". In server mode, the adapter continuously polls for incoming messages and passes them to the other adapters as they are available.
5. When the originating adapter completes its execution, the EAI Server Component terminates.

In the Siebel 99.1 implementation, adapters are single threaded and execute all methods synchronously. As a result, passing a message through the adapters is just a series of nested execute() method calls. Status and response messages are passed back up as the execute() methods return. For example, consider an integration involving the Peoplesoft adapter (configured to receive messages from Peoplesoft), the Data Transformation Engine adapter, and the EIM adapter. Once the EAI Server Component has connected all the adapters, it will call the start() method of the Peoplesoft adapter. This adapter retrieves a batch of messages from Peoplesoft and passes them to DTE by calling the execute() method of DTE. DTE transforms the messages to Siebel objects and calls the execute() method of the EIM adapter, providing the new messages as a parameter to the method. The EIM adapter runs EIM to import the data into Siebel and then returns status information. Control is the passed back to DTE, which performs any necessary cleanup and then returns its status. The Peoplesoft adapter can then update its status and either return back to the EAI component or wait for more messages. Below is a diagram that demonstrates the nesting of adapter method calls when a message has reached the EIM adapter.



Nesting of adapter calls during a message transfer

2.2.1 Error Handling

For 99.1, the transfer of each message batch will be treated as a single logical transaction. If an adapter encounters an error when processing a message batch, it will roll back any changes it has made and return an error to its caller. Each caller will, in turn, undo all changes related to the current message batch and return an error status. Upon receiving an error from the originating adapter, the EAI Server Component will make the error known to the user and terminate the component. Any messages fetched from an external application must be stored locally or pushed back to the application (perhaps just by changing their status in the application's staging tables).

Each adapter that originates messages will have the capability to break a batch of messages into smaller subsets. If an error cannot initially be isolated to a single message, it will be possible to process messages one at a time until the error occurs.

In a future release, we will add the ability determine success or failure at the individual record level. Keys in the destination adapter will be mapped back to the corresponding keys in the source adapter. Each source row will be marked as either "successful" or "error". See the "Deferred Features" chapter for more details.

2.3 Future Architectural Enhancements

In Siebel 2000, we may want to extend this architecture in the following ways:

- Make the adapters thread safe and change the interface to allow adapter methods to be called asynchronously. This will make it possible to "pipeline" messages through a series of adapters and to have more than one originating adapter in a process.
- Allow an adapter to have multiple adapters connected to its input and output ports. It may be advantageous to add a message filtering mechanism to the outputs (a message is sent to an output adapter only if it satisfies a certain predicate).

- As mentioned previously, add in-memory message representations along with utility functions to convert between representations. Memory representations may be more efficient, particularly if DTE is enhanced to perform transformations outside of the server database. In addition, we may want to pass these messages between processes or store them in a persistent queue.

3 Implementation Overview

3.1 EIM Adapter

3.1.1 Overview

This message describes a method for using the Siebel Remote transaction log or Workflow Manager to track Siebel transactions that need to be sent to external systems (e.g. PeopleSoft).

3.1.2 Proposed Solution

Create an "outbound queue" table (S_EAI_OUT_SYNC) to store a list of items (e.g. Accounts, Orders, etc) that have been sent to the external system. The table has a row for every item that can be sent to the external system. For each item, the table stores the last "update time" (denoted by a Siebel Remote txn id) that the item was updated and a "sent time" that stores the last time EAI sent the item to the external system. Workflow Manager or a new server component that reads the Siebel Remote transaction log writes rows into this "outbound queue" table. The EAI outbound flow uses the "outbound queue" table to identify the items that need to be sent to the external system. For example, EIM uses a where clause to find all accounts that are in the "outbound queue" table that have been updated since the last "sent time". After sending the item to the external system, the EAI outbound flow updates the last "sent time".

3.1.3 Outbound Queue Table: S_EAI_OUT_SYNC

This table keeps track of the object instances (e.g. accounts, orders, etc) that need to be extracted from Siebel and sent to the external system.

Column	Data Type	Foreign Key Table	Description
OBJECT_ID			object (e.g. Account, Contact, Order)
PR_TBL_ROW_ID			ROW_ID of the primary table in the object
UPD_TXN_ID			last txn id that changed the object
SYNC_TXN_ID			max txn id in S_DOCK_TXN_LOG when this object was last sent to the external system

3.1.4 Writers for the Outbound Queue Table

TxnProcessor reads TxnProc .dx files and writes rows into the "outbound queue table".

1. Read txn from TxnProc .dx file.
2. If a row for the current txn's <object id, pr_tbl_row_id> already exists, then update the TXN_ID to the curent txn's txn id.
3. Else, insert a new row for the current txn's <object id, pr_tbl_row_id, txn_id>
4. commit

Alternatively, we can use Workflow Manager to populate the "outbound queue table".

3.1.5 Readers for the Outbound Queue Table

Outbound EAI flow uses this table to identify object instances for EIM to export to the interface tables.

For a given object:

1. Get max txn id in S_DOCK_TXN_LOG
2. Lock table S_EAI_OUT_SYNC in exclusive mode (to prevent deadlocks with the writer)
3. Update all rows where UPD_TXN_ID > SYNC_TXN_ID
4. Get all rows where UPD_TXN_ID = SYNC_TXN_ID and drive EIM export
5. Do other EAI steps (DTE, push into peoplesoft staging tables, etc)
6. commit

3.1.6 Notes/Issues

- We can also add columns to this table to store mapping table between Siebel and the external system's IDs instead of storing INTEGRATION_ID in Siebel base tables.
- We can build a screen on this table to show the last time a object was synchronized.
- To make EIM export easier, we could add a flag column in each Siebel base table to tag rows that EIM needs to export. This will simplify the EIM .ifb files (get all rows where <eim_tag_flg> = 'Y'>. But requires a non-unique index!

3.2 Data Transformation Engine

We wish to build a declarative tool integrated with Siebel's repository and tools for mapping data between Siebel's application interfaces (Interface Tables and Business Components) and other data representations (e.g., PeopleSoft staging tables or SAP IDOC messages). The design of this tool will be optimized for heterogeneous object synchronization.

The DTE schema is flexible about the types of mappings supported. There will be additional validation performed to ensure that a given mapping is well-formed and will work for its intended use. Eventually, this validation could be performed dynamically by a tools interface that allows only well-formed mappings to be created.

3.2.1 Mapping Procedure

The procedure for creating a map (from the user's point of view) is as follows:

1. Create an integration object map in tools and specify the two objects to be mapped. In addition, specify whether the map is to be bi-directional, Siebel-to-External, or External-to-Siebel.
2. Determine the mappings between components in the Siebel object and components in the external object. Two components are mapped to each other if they contain fields that will be mapped to each other. Create component mappings in tools that reflect these mappings. The tools mappings should obey the following rules:
   - Each mapping should include as few components as possible – only those necessary to perform the desired field mappings.
   - In a component map, only one component from each Integration Object can have a parent not in the map. The other components in the map must descend from these two components. The two components without a parent in the map are called *root components.*
   - There will be additional restrictions for bi-directional maps.
   - For each non-root component in the component maps, specify how the component can be merged into its parent. Currently, two merge types are permitted: de-normalization and pivoting. De-normalization appends the key of the child component to the parent component. Pivoting converts records (rows) in the child to fields (columns) in the parent. For pivoting, a *pivot key* must also be specified. This is the field in the child component that determines which column a row should be mapped to.

3. A filter expression may be specified for each component in a component mapping. The filter expression selects a subset of the records in that component.
4. Define field-level mappings for each component map. A field mapping includes the following elements:

   Direction

   This is the direction of the mapping – either Bi-directional, Siebel-to-External, or External-to-Siebel. If an object map is unidirectional, the field mappings only need to be unidirectional. If the object map is bi-directional, fields can be mapped through either a bi-directional map or two unidirectional maps (one in each direction).

   Siebel Field

   This is a field from one of the Siebel components in the component map. Whether it is a source or destination (or both) depends on the value of *direction.*

   External Field

   This is a field from one of the External components in the component map.

   Calculated Value

   If the field mapping is unidirectional and it depends on more than one source field, then an expression to calculate the destination field's value based on the source fields is entered here. For Siebel-to-External mappings, *External Field* is set, but *Siebel Field* is left empty. For External-to-Siebel mappings, *Siebel Field* is set and *External Field* is left empty.

   Pivot Key Value

   If the field being mapped is part of a pivot transformation (converting between records and fields), then the value of the pivot key for this mapping is specified.

3.2.2 DTE Theoretical Model

Different levels of validation (and thus classes of maps) are possible. For example, some transformations cannot be used when implementing a bi-directional map, but are perfectly valid for unidirectional maps. In general, the types of transformations possible depend on the direction of synchronization (Siebel-to-External, External-to-Siebel, or both) and the capabilities/requirements of the application adapters. The model used by DTE has the following advantages:

- Mapping flexibility can be tailored to the capabilities of each application adapter. The alternatives are to restrict mappings to the requirements of the least-flexible adapter or to store an extra copy of each synchronized object in less-capable adapters.
- Partitions (see below) make it easy to determine which components from the source are needed to build a destination component. Otherwise, the entire Integration Object must be sent when a change is made in one of its components. [Siebel could actually do a field-dependency analysis on its outbound data, but other applications do not provide field-level change information].
- Restricted mapping models allow DTE to ensure that maps are well-formed and that they will perform as expected. For example, DTE will not allow the definition of maps that could result in SQL-level runtime errors (such as unique key constraints).
- Bi-directional mappings can be defined and validated. This avoids the need to define two one-directional maps when bi-directional synchronization is needed. In addition, it allows Siebel to define a single map that will work in either direction and then let the customer decide the direction without changing the map.

Initially, the most restrictive validation rules will be used. After we better understand the capabilities of application adapters and the requirements for synchronization, restrictions will be relaxed for some classes of maps. The restricted mapping class that we will define for Siebel 99.1 will have the following properties:

- Mappings are made between two integration objects. They can be for a single direction or bi-directional.
- The types of transformations allowed will be restricted to those that can preserve the source object's component level keys. This is necessary for bi-directional synchronizations and for the automatic generation of component-level Create/Update/Delete fields. The permitted component-level transformations are:

1. De-normalization of parent and child components by concatenating the keys of the parent and child.
2. Converting rows in a child component to columns in the parent component using a field in the child component to determine the column for each row. This is called "pivoting".
3. Mapping only those records (of a given component type) that satisfy a given predicate. This is called "filtering".
4. Mapping only a subset of an integration object's components.

- The key preservation requirement also restricts the form of component-to-component mappings. One to one component mappings, one to many mappings, and many to one mappings are permitted. Many to many component mappings are not allowed.

From the theoretical point of view, legal mappings are defined using the following procedure:

1. Construct an acyclic, directed graph based on the Siebel Integration Object where the nodes of the graph are the object's components and the edges of the graph are the parent-child relationships.
2. Construct an acyclic, directed graph based on the ERP Integration Object.
3. Make the two graphs isomorphic by removing selected edges in each graph and merging the nodes that were previously connected by these edges. Each time a child is merged into its parent, the merge is labeled with an operation, either "de-normalization" or "pivot". The nodes in the new graph are called *partitions*, since they are formed by partitioning the original graph. The new graphs based on these partitions are called *partition graphs.*
4. Label each partition in the two graphs either "One" or "Many". Partitions are label "one" if they include exactly one node from the original graph and "many" if they include more than one node from the original graph.
5. Associate each node of the Siebel partition graph with a node of the ERP partition graph. Each node can be a part of only one association and the two nodes must be locally equivalent. Two nodes are locally equivalent if the sub-graphs below them are isomorphic. In addition to being equivalent, associated nodes cannot both be labeled "many".
6. Field mappings can only be made between associated nodes of the partition graphs.

The restricted model can be relaxed in several ways. The follow tables shows the implications of different rule changes:

Mapping Rules	Implications for directionality of maps	Implications for adapter requirements
Restricted rules	Maps can be bi-directional if they are at the field level.	In general, snapshots can be provided at the object level or the partition level. Create/Update/Delete information at the component level can be provided by DTE.
Many to many partition associations are allowed.	Mappings can be single-directional only.	Snapshots can be provided at the object level or the partition level. Create/Update/Delete information can only be provided at the partition level.
Partitions can overlap.	Mappings can be bi-directional.	Snapshots must include all partitions that include a changed component (or just use an object snapshot). Create/Update/Delete information can only be provided at the partition level.
New merge type: aggregation of child into parent	Mappings can be single-directional only.	Snapshots can be provided at the object level or the partition level. Create/Update/Delete information at the component level can be provided by DTE.

3.3 DTE Generic Expression Language

The following BNF grammar defines the language for DTE calculated expressions.

```
scalar_exp:
                scalar_exp '+' scalar_exp
        |       scalar_exp '-' scalar_exp
        |       scalar_exp '*' scalar_exp
        |       scalar_exp '/' scalar_exp
        |       '+' scalar_exp
        |       '-' scalar_exp
        |       literal
        |       column_ref
        |       function_ref
        |       '(' scalar_exp ')'
        |       case_statement

scalar_exp_commalist:
                scalar_exp
        |       scalar_exp_commalist ',' scalar_exp


case_statement:
                CASE scalar_exp when_exp_list END
        |       CASE scalar_exp when_exp_list  else_clause

else_clause:
                ELSE scalar_exp END

when_exp:
                WHEN scalar_exp THEN scalar_exp

when_exp_list:
                when_exp
        |       when_exp_list when_exp


literal:
                STRING
        |       INTNUM
        |       APPROXNUM

column_ref:
                NAME
        |       NAME '.' NAME

function_ref:
                AMMSC '(' '*' ')'


        |       AMMSC '(' DISTINCT column_ref ')'

        |       AMMSC '(' ALL scalar_exp ')'

        |       AMMSC '(' scalar_exp_commalist ')'

        |       NAME '('  scalar_exp_commalist ')'
```

AMMSC is defined as the aggregate functions SUM, MIN, MAX, AVG (JFISCHER:VERIFY)

The tokens STRING, NAME, INTNUM, APPROXNUM are defined as:

```
NAME:           [A-Za-z][A-Za-z0-9_]*
STRING:         '[^'\n]*'
INTNUM:         [0-9]+              |
                [0-9]+"."[0-9]*     |
                "."[0-9]*
APPROXNUM:      [0-9]+[eE][+-]?[0-9]+            |
                [0-9]+"."[0-9]*[eE][+-]?[0-9]+   |
                "."[0-9]*[eE][+-]?[0-9]+
```

For definitions and examples of the symbols used in the above regular expressions see a Lex and Yacc book, such as "lex and Yacc" by Levine, Mason, and Brown published by O'Reilly Associates

3.3.1 Examples of Parse Trees

In this section we diagram the parse trees for several expressions. Leaves of the parse trees represent tokens, while non-leaves represent rules. We do not create non-leaf nodes for each matching rule.

Example 1:
Expression: 4, foo() // to be added later

3.3.2 DTE Expression Language Functions

The following functions will be provided by the expression language:

- Substring()
- Charindex()
- Len()
- Replace()
- Isnull()
- lookup(<lookup_type>, <value>)

3.4 PeopleSoft Adapter

3.4.1 PSFT Adaptor APIs (pseudo-code design)

3.4.1.1 RetCode sebl_inbound(IntegObj)

```
{
    // setup db connection
    APP = {PSFT}
    conn(APP)  = Get the connection handle to APP DB
    stmt(APP)  = Allocate a SQL statement obj on conn(APP)

    listOfBatchProcessed = { }

    // call Integ MetaData API to get
    //  (1) PSFT staging table name for this IntegObj
    //  (2) all the column names for this staging table
    // in order to generate the SQLstmt

    // get the msg content by joining PSFT's MSG queue and staging tables
    SQLstmt =
    select TRANS_PROGRAM, TRANSACTION_CODE,
           CIP_MSG_NUM, CIP_MSG_SEQ_NUM, CIP_MSG_ACTN,
           { all the application columns }
    From PS_CIP_OUT_SUBQ  Q,  PS_CIP_IntegObj_OT S
    Where    Q.CIP_SUBSCRIBER_ID = 'Siebel Systems'
           And Q.CIP_TXN_FLAG = 'Outbound'
           And Q.CIP_MSG_STAT in ('NEW', 'RESUBMITTED')
           And Q.TRANS_PROGRAM = IntegObj )
           And Q.TRANS_PROGRAM = S.TRANS_PROGRAM
           And Q.TRANSACTION_CODE = S.TRANSACTION_CODE
           And Q.CIP_MSG_NUM = S.CIP_MSG_NUM
           And Q.CIP_MSG_SEQ_NUM = S.CIP_MSG_SEQ_NUM
    Order by Q. CIP_MSG_NUM, Q.CIP_MSG_SEQ_NUM

    numRows = SQLstmt.execute

    // trap DB error here

    // do row-size batching to avoid processing e.g. 1 Million rows,  at once.
    If numRows > PredefinedBatchSize
          Fetch the first PredefinedBatchSize rows
          numRows = PredefinedBatchSize
    Else
          Fetch all the rows

    BatchNumber = generate an unique batch number

    // ??? open issue here:  snapshot or delta of IntegObj
    Loop over numRows
    If  CIP_MSG_ACTN = 'Update'
          RetCode = reconstruct_snapshot(IntegObj, CIP_MSG_NUM)
    // this function needs to query PSFT Application Table for this particular instance of IntegObj
    // and reconstruct this CIP_MSG and save it to memory
    // it is still not finally decided yet whether PSFT side or SEBL side will do this ???

    // move data from memory to Siebel staging table (inbound)
    RetCode = insert_sebl_stgtbl_in (IntegObj, numRows, BufferAddress, BatchNumber)

    If RetCode = OK
        update PS_CIP_OUT_SUBQ set CIP_MSG_STAT='Picked' for those picked rows
        ListOfBatchProcessed.add(BatchNumber)

        // commit change in PSFT database

    Else
        LogError "fail to insert numRows of IntegObj into Siebel Staging Table"


    // continue to process next batch-size if any


If listOfBatchProcessed.is_empty()
    LogInfo "No MSG found"
    return
Else
    RetCode = StartDTE(ListOfBatchProcessed)

If RetCode = OK,
    update PS_CIP_OUT_SUBQ set CIP_MSG_STAT='Success' where
        CIP_MSG_STAT = 'Picked'
Else
    If RetCode indicates Functional Failure
    // e.g. EIM fails to load Dictionary/mapping, DTE or EIM fail to start
        update PS_CIP_OUT_SUBQ set CIP_MSG_STAT='New' where
            CIP_MSG_STAT = 'Picked'
    Else
    // e.g. data failure, PSFT needs to inspect data quality/integrity and resubmit
        update PS_CIP_OUT_SUBQ set CIP_MSG_STAT='Error' where
            CIP_MSG_STAT = 'Picked'

// commit status update in PSFT DB

// free SQLstmt
// free PSFT connection

}
```

3.4.1.2 *RetCode insert_sebl_stgtbl_in (IntegObj, numRows, BufferAddress, BatchNumber)*

```
{
// setup db connection
APP = {SEBL}
conn(APP) = Get the connection handle to APP DB
stmt(APP) = Allocate a SQL statement obj on conn(APP)
// call Integ MetaData API to get
//  (1) PSFT staging table name for this IntegObj
//  (2) all the column names for this staging table
// in order to generate the SQLstmt

// get Sebl staging table name: S_STG_PS_IntegObj_IN

// get the ROW_Id for each INTEG_COMP that belongs to the given IntegObj
// saved in a local array/map

// perform in-memory manipulations to
// (1) trim PSFT application columns because spaces are used to make it NOT NULL
// (2) update EAI_INTEG_COMP column
// (3) filter out unmapped PSFT app columns

SQLstmt = Insert into S_STG_PS_IntegObj_IN(
        EAI_BATCH_NUM, EAI_STAT, EAI_INTEG_COMP,
        TRANS_PROGRAM, TRANSACTION_CODE,
        CIP_MSG_NUM, CIP_MSG_SEQ_NUM, CIP_MSG_ACTN,
        { only the mapped PSFT app. columns } )
        values(BatchNumber,'New', INTEG_COMP_ROWID,
        ........................)

SQLstmt.execute()

// trap DB error here

// commit the insert

If OK
        return OK
Else
        Return ErrCode

}
```

3.4.1.3 *retCode reconstruct_snapshot(IntegObj, CIP_MSG_NUM)*

```
{

// to be designed

}
```

3.4.2 Scripts to populate S_INTEG_OBJ table for PSFT

```
insert into S_INTEG_OBJ(NAME, EXT_NAME, BASE_OBJ_TYPE, ADAPTER_INTO)
        values('PSFT CIP: Customer', 'PS_CIP_CUSTOMER_OT', 'EAI_PS_CUST', 'PSFT')
insert into S_INTEG_OBJ(NAME, EXT_NAME, BASE_OBJ_TYPE, ADAPTER_INTO)
        values('PSFT CIP: Employee', 'PS_CIP_STG_EMPLOYEE', 'EAI_PS_EMP', 'PSFT')
insert into S_INTEG_OBJ(NAME, EXT_NAME, BASE_OBJ_TYPE, ADAPTER_INTO)
        values('PSFT CIP: Product', 'PS_CIP_STG_PRODDET', 'EAI_PS_PROD', 'PSFT')
insert into S_INTEG_OBJ(NAME, EXT_NAME, BASE_OBJ_TYPE, ADAPTER_INTO)
        values('PSFT CIP: Price List', '??????', 'EAI_PS_PRLST', 'PSFT')
```

3.5 SAP Adapter

3.5.1 Introduction to IDOCs

The scope of this project is limited to inbound processing of master data for customers and products only. The SAP adapter relies on the IDOC (immediate document) communication mechanism provided by SAP. The IDOC types to which the scope of this project is limited are:

- DEBMAS02 (master data for customers)
- MATMAS02 (master data for products)

An IDOC is essentially a text document. Each IDOC is divided into logical segments, and each segment contains a specific number of fields. Each IDOC type has a specific grammar for the segements within it. An example of segment grammar for the DEBMAS02 IDOC type:

```
E1KNA1M : Master customer master basic data (KNA1)
Status : Mandatory
  E1KNA1H : Master customer master basic data: texts, header
  Status : Optional
    E1KNA1L : Master customer master: text lines
    Status : Optional
  E1KNVVM : Master customer master sales data (KNVV)
  Status : Optional
:
:
```

Within a parent segment, there could be a repetition of child segments, however the order of individual children segments within a parent segment must be maintained.

Each segment must contain all the fields specific to that segment. That is, there are no optional fields. All fields are fixed length. So, within a segment each field has an offset associated with it. If there is no data for a particular field, SAP generates spaces for those fields in the IDOC. There are no field separator characters between fields in a segment. An example of the list of fields in the segment E1KNA1M within the DEBMAS02 IDOC above is:

```
E1KNA1M : Master customer master basic data (KNA1)

  MSGFN : Function
  internal data type : CHAR
  internal length : 000003
  Position in structure : Offset : 0055. external length :
  000003

  KUNNR : Customer number
  internal data type : CHAR
  internal length : 000010
  Position in structure : Offset : 0058. external length :
  000010
:
:
```

When multiple records of the master data are modified (creation, deletion, update), SAP generates one IDOC for each record. It does not batch the information about multiple records into a single IDOC.

A mapping between IDOC terminology and the integration object terminology is as follows:

- An IDOC corresponds to an integration object
- An IDOC segment corresponds to an integration component
- An IDOC field corresponds to an integration field

3.5.2 Scripts to populate integration objects and creating staging tables

A design decision was taken to allocate one staging table for each IDOC segment.

- (explain rationale)

SAP provides meta-data files called IDO files describing the grammar for each IDOC type.
Procedure to generate an IDO file: Transaction WEDI. Documentation → IDOC type (parser).

An in-house tool is built (in JAVA) to automatically generate scripts for creating staging tables and populating integration meta-data tables, given an IDO file as input. The tool resides in \\APPINT0\EAI Resources\SAP Resources\IDOC\Idoc MetaData\IdoParser.ZIP

- (document few idiosyncrasies of IDO files)
- (document some hardcodings, use of INACTIVE_FLG to include all segments and all fields in integration meta-data tables, etc.)

Other than the four EAI specific columns (described in the sub-component interfaces later in the document), SAP staging tables do not contain any other special columns. All other columns in a staging table correspond to the fields for that IDOC segment.

Use of the column ADAPTER_INFO:

- Integration Object: Unused
- Integration Component: Level of an IDOC segment in the parent-child hierarchy (for information only. Not used in the code right now.)
- Integration Field: Offset of a field within the IDOC segment

3.5.3 Communication of the SAP adapter with SAP

RFC (SAP's proprietary API) for communication is used for connecting the SAP adapter with gateway on a SAP system.

- (document procedure to generate stub RFC code from SAP gui)

RfcReceive is a blocking function.

- (explain how for each IDOC segment, the registered function with RfcReceive gets woken up each time. Need to process an IDOC segment one at a time.)
- Insert pseudo-code for the RFC server mechanism used by the SAP adapter.

3.5.4 IDOC parsing

Use of the ADAPTER_INFO column in the S_INT_FIELD makes parsing a character buffer containing an IDOC segment trivial.
March down the character buffer based on the LENGTH information of individual fields. Query the S_INT_FIELD table with a WHERE ADAPTER_INFO = "…" clause to determine the name of the field and its length.
Use the length information to determine how much to march forward.
Generate an INSERT INTO staging_table statement based on the name of the column retrieved from S_INT_FIELD.

- Insert pseudo-code that illustrates this.

3.5.5 Buffering of IDOCs

Use of the SAP adapter decided batch number to batch the IDOCs and avoid inserting one row at a time into the staging tables.

- Insert pseudo-code to allocate batch numbers and do the buffering Use of the UtlData function(s) to insert multiple rows at a time.

3.5.6 SAP related open issues

- Inability to resend a lost IDOC.
- Inability to programatically (using RFC) query the IDOC queue for an SAP system to find out which IDOCs the adapter was supposed to get when it was down.
- Error recovery when a partial IDOC is received and the adapter crashes
- Error recovery when an IDOC segment is inserted into the staging table, but the adapter crashes before committing.

4 Summary of Affected Components

4.1 Data Model

4.1.1 Repository Tables

4.1.1.1 New Tables

- *S_INT_OBJ* – Integration Object definitions
- *S_INT_COMP* – Integration component definitions
- *S_INT_FIELD* – Integration component field definitions
- *S_INT_OBJMAP* – Top-level table for DTE Maps
- *S_INT_COMPMAP* – DTE component level mappings
- *S_INT_CMAPCMP* – Component in a component level mapping
- *S_INT_FLDMAP* – Field level mappings

4.1.2 Application Tables

4.1.2.1 New Tables

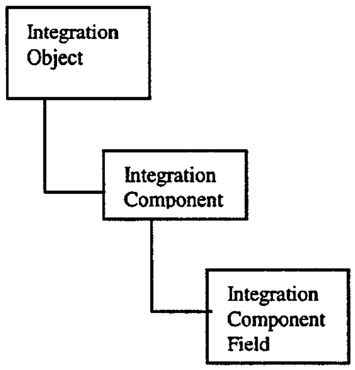
- *S_EAI_APPLVER* – Information about Siebel and External applications
- *S_EAI_ADAPTER* – Adapters
- *S_EAI_APPL_ADPT* – Intersection table between applications and adapters
- *S_EAI_FLOW* – Information about adapter sequences that EAI component can execute
- *S_EAI_ADPT_FLOW* – Elements (adapters) that make up a sequence.
- *S_EAI_LOOKUPMAP* – Table for performing lookup mappings

5 User Interface Specification

5.1 Tools Interface

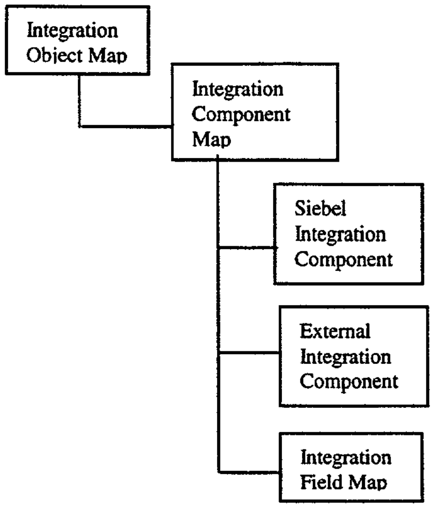
5.1.1 Tools Interface for Integration Object Definitions

5.1.1.1 Explorer View



5.1.2 Tools Interface for Data Transformation Engine

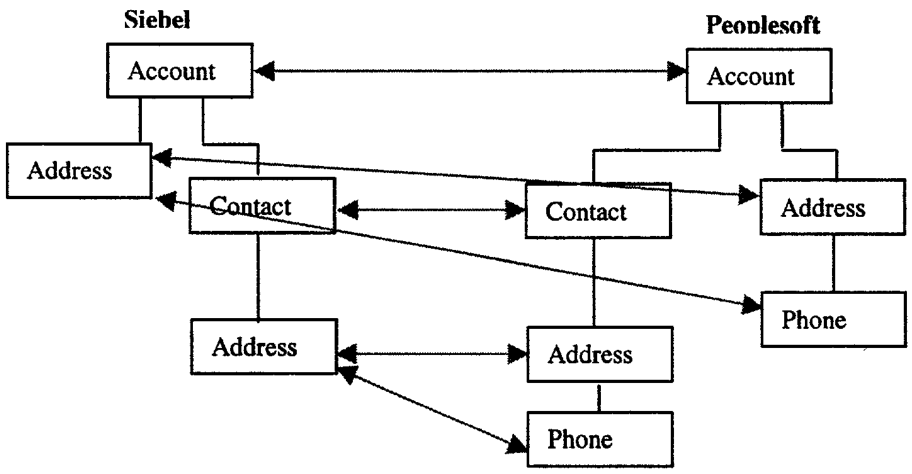
5.1.2.1 Explorer View


Integration Object Map
Integration Component Map
Siebel Integration Component
External Integration Component
Integration Field Map

5.1.2.2 DTE Tools Interface for Siebel 2000

We would like to have a nicer GUI interface for Siebel 2000. The steps of defining a map with this new interface would be:

1. Pick an Integration Object from Siebel and from the ERP application
2. Pick the Siebel and ERP adapters (the capabilities of these adapters influence what transforms will be allowed).
3. Determine whether the mapping will be Siebel-to-External, External-to-Siebel, or bi-directional.
4. Tools then shows a screen with a depiction of each Integration Object as a tree. The user defines component mappings by drawing arrows between the components of the two trees. Here's an example:

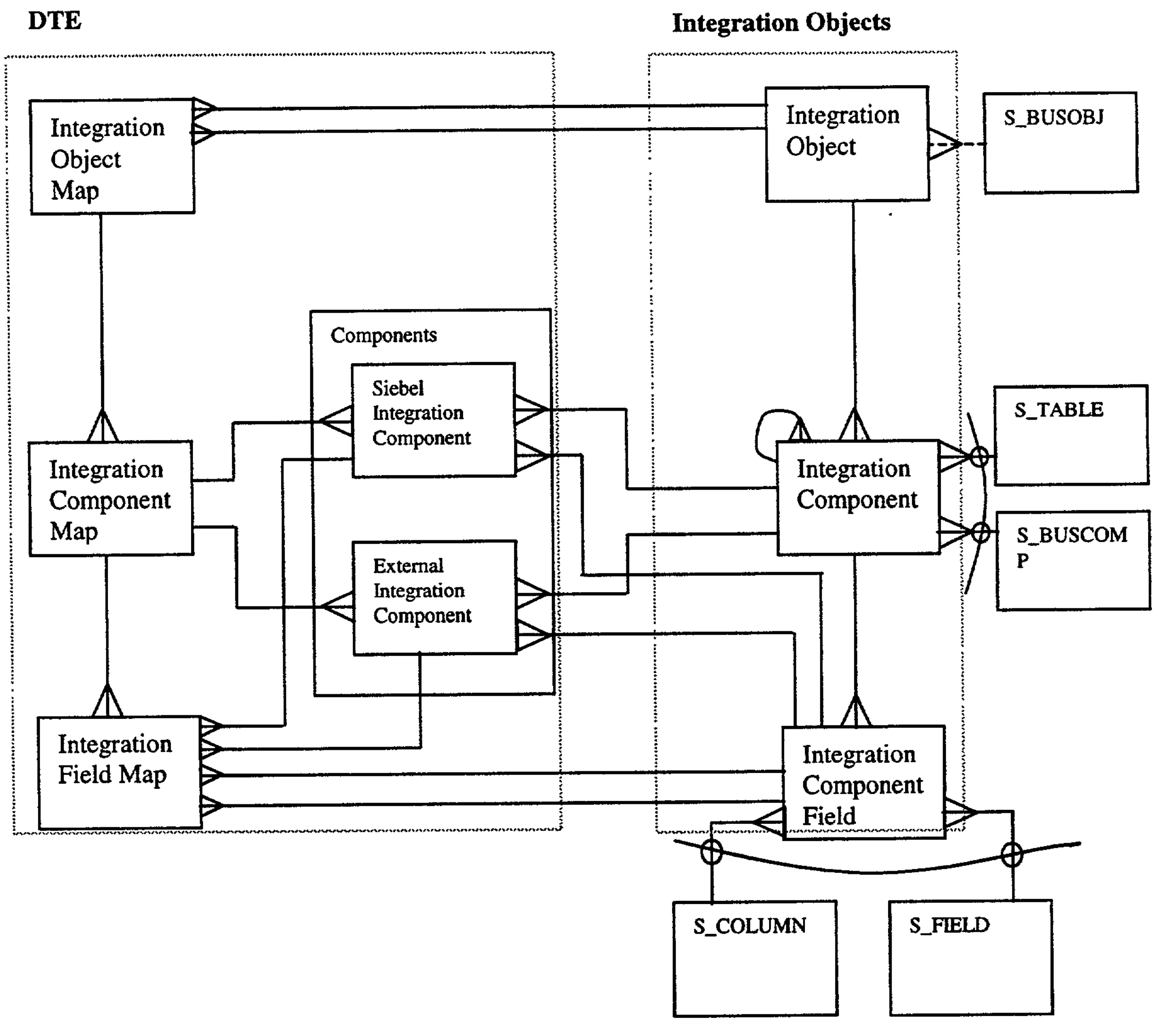


5. Next, tools calculates the partitions based on these mappings. If any of the mappings violate the partitioning rules for this map, then tools shows those arrows in red, provides an explanation, and allows the user to change them. After the mappings have been accepted, tools goes through a sequence of dialogs where it asks the user how each child object should be merged into its parent. The user answers either de-normalization or pivot to each question.
6. Tools then inserts the valid component mappings into the schema.
7. Finally, the user enters field level mappings (perhaps using a drag-and-drop GUI).

6 Repository Data Model Specification

This section lists additions/changes to the Siebel repository tables.

6.1 E-R Diagram for DTE and Integration Objects



6.2 Schema for Integration Object Metadata

6.2.1 S_INT_OBJ (Integration Object)

This is the base table for storing integration object definitions. It is a new "top level" repository object. An integration object is a hierarchical object defined on top of a set of tables or business components. The entities within an integration object are called integration components. It can be used as a source or destination in a DTE map. Integration Component Adapters (e.g. for SAP or PeopleSoft) provide these object definitions based upon metadata from the external application.

Columns for logical definition

Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The user name given to a integration object.
COMMENTS	Varchar:250	Comments		

Columns for physical definition

Column	Data Type	Field Name	Foreign Key Table	Usage
EXT_NAME	Varchar:75	External Name		External name of this object (used by adapter).
BASE_OBJ_TYPE	Varchar:30	Base Object Type		Pick-list: "Table" (in Siebel 2000, BusComp will be added)
BUS_OBJ_NAME	Varchar:75	Business Object Name	S_BUSOBJ (by name)	If the base object type is Business Component, then this is the name of a Business Object definition
ADAPTER_INFO	Varchar:250	Adapter Information		Adapter-specific information.

6.2.2 S_INT_COMP (Integration Component)

This table stores the definitions of the components (entities) that make up an integration object. A reference is made to the physical object (table or BusComp) upon which the component definition is based.

Columns for logical definition

Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The user name given to the component.
INT_OBJ_ID	Varchar:15	Integration Object	S_INT_OBJECT	Parent object type definition for this component.
PAR_INT_COMP	Varchar:75	Parent Integration Component	S_INT_COMP (by Name)	Parent component of this component. Null, if this is the root component for an integration object. The pick list should be restricted to components in the same integration object.
COMMENTS	Varchar:250	Comments		

Columns for physical definition

Column	Data Type	Field Name	Foreign Key Table	Usage
EXT_NAME	Varchar:75	External Name		External name of this component (used by adapter).
SHARE_FLG	Boolean	Share Flag		If this flag is true, then the component is sharing a physical table with other components of the same integration object. The column *EAI_INTGR_COMP_ID* indicates which rows belong to this component.
TBL_NAME	Varchar:75	Table Name	S_TABLE (by name)	If the base object type in S_INTGR_OBJECT is "table", then this column contains the name of the table that is the basis of the component's definition.
BC_NAME	Varchar:75	Business Component Name	S_BUSCOMP (by name)	If the base object type in S_INTGR_OBJECT is "business component", then this column contains the name of the BusComp that is the basis of the component's definition.
ADAPTER_INFO	Varchar:250	Adapter Information		Adapter-specific information.

6.2.3 S_INT_FIELD (Integration Component Field)

This table specifies one field of an integration component, referencing the definition of its base object (column or business component field).

Columns for logical definition

Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The name of this field
INT_COMP_ID	Varchar:15	Integration Component	S_INT_COMP	Pointer to parent component of this field
PHYS_DATA_TYPE	Varchar:30	Physical Data Type		The physical data type of the field (same as Siebel column types).
LENGTH	Number	Length		Length of field (for character data types).
PREC_NUM	Number	Precision		
SCALE	Number	Scale		
REQUIRED_FLG	Boolean	Required Flag		True if this field must be present.
USER_VISBL_FLG	Boolean	User Visible Flag		True if the field is visible to the DTE user.
FIELD_TYPE_CD	Varchar:30	Field Type		Pick-list: "System(Siebel)", System(External)", "Data"
KEY_SEQ_NUM	Number	Key Sequence		If the field is part of the key of the component, the position of the field within the key.
PAR_KEY_SEQ_NUM	Number	Parent Key Sequence		If the field is part of the parent key of the component, the position of the field within the key.
COMMENTS	Varchar:250	Comments		

Columns for physical definition

Column	Data Type	Field Name	Foreign Key Table	Usage
EXT_NAME	Varchar:75	External Name		External name of this field (used by adapter).
COL_NAME	Varchar:75	Column Name	S_COLUMN (by name)	If the base object type in S_INTGR_OBJECT is "table", then this column contains the name of the column that is the basis of the field's definition. Pick-list should only show the columns from the table specified in the integration component.
BCFLD_NAME	Varchar:75	Business Component Field Name	S_FIELD (by name)	If the base object type in S_INTGR_OBJECT is "business component", then this column contains the name of the BusComp field that is the basis of the field's definition. Pick-list should only show the fields belonging to the BusComp specified in the integration component.
ADAPTER_INFO	Varchar:250	Adapter Information		Adapter-specific information.

6.3 Schema for Data Transformation Engine

6.3.1 S_INT_OBJMAP (Integration Object Map)

This table is the root of a map definition for the Data Transformation Engine. It is a "top level" repository object. It links a Siebel integration object and an External integration object.

Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The name of this map
SEBL_INTOBJ_NAME	Varchar:75	Siebel Integration Object Name	S_INT_OBJECT (by name)	Name of Siebel Integration Object
EXT_INTOBJ_NAME	Varchar:75	External Integration Object Name	S_INT_OBJECT (by name)	Name of External Integration Object
DIRECTION_CD	Varchar:30	Direction		Pick-list: "Siebel to External", "External to Siebel", and "Bi-directional"
AUTO_TRUNCATE_FLG	Boolean	Auto-truncation flag		True, if DTE should automatically truncate character fields that are longer than their destination column.
COMMENTS	Varchar:250	Comments		

6.3.2 S_INT_COMPMAP (Integration Component Map)

This table specifies the mapping between two sets of components.

Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The name of this component mapping.
INT_OBJMAP_ID	Varchar:15	Object Map	S_INT_OBJMAP	The object map that this component map is a part of.
COMMENTS	Varchar:250	Comments		

6.3.3 S_INT_SEBLCMP (Siebel Integration Component)

This table specifies the Siebel Integration Components in a component map

Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The name of this component when used within the mapping. Basically, an alias for the component.
INT_COMPMAP_ID	Varchar:15	Component Map	S_INT_COMPMAP	The component map that this component is a part of.
INT_COMP_NAME	Varchar:75	Integration Component Name	S_INT_COMP (by name)	The name of a component within the integration object to be included in the component mapping. The pick-list for this field should only show the components of the relevant Siebel integration object.
MAP_KEYS_FLG	Boolean	Map Keys Flag		If true, map the keys for this component.
FILTER	Varchar:2000	Filter		Filter expression to select a subset of this component's records.
MERGE_TYPE_CD	Varchar:30	Merge Type		Pick-list: "None", "De-normalization", "Pivot"
PIVOT_KEY_FIELD	Varchar:75	Pivot Key Field	S_INT_FIELD (by name)	Name of field to be used as pivot key if merge_type = pivot. Pick-list should be restricted to the fields within the specified integration component.
COMMENTS	Varchar:250	Comments		

6.3.4 S_INT_EXTCMP (External Integration Component)

This table specifies the Siebel Integration Components in a component map

Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The name of this component when used within the mapping. Basically, an alias for the component.
INT_COMPMAP_ID	Varchar:15	Component Map	S_INT_COMPMAP	The component map that this component is a part of.
INT_COMP_NAME	Varchar:75	Integration Component Name	S_INT_COMP (by name)	The name of a component within the integration object to be included in the component mapping. The pick-list for this field should only show the components of the relevant external integration object.
MAP_KEYS_FLG	Boolean	Map Keys Flag		If true, map the keys for this component.
FILTER	Varchar:2000	Filter		Filter expression to select a subset of this component's records.
MERGE_TYPE_CD	Varchar:30	Merge Type		Pick-list: "None", "De-normalization", "Pivot"
PIVOT_KEY_FIELD	Varchar:75	Pivot Key Field	S_INT_FIELD (by name)	Name of field to be used as pivot key if merge_type = pivot. Pick-list should be restricted to the fields within the specified integration component.
COMMENTS	Varchar:250	Comments		

6.3.5 S_INT_FLDMAP (Integration Field Map)

This table specifies a field-level mapping.

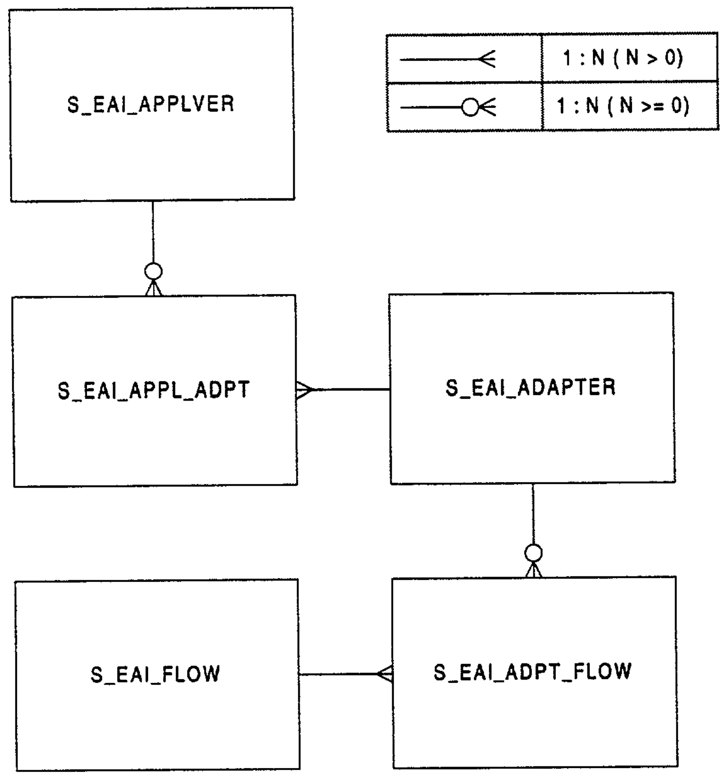
Column	Data Type	Field Name	Foreign Key Table	Usage
NAME	Varchar:75	Name		The name for this mapping.
INT_COMPMAP_ID	Varchar:15	Component Map	S_INT_COMPMAP	The parent component map for this field.
DIRECTION_CD	Varchar:30	Direction		Pick-list: "Siebel to External", "External to Siebel", and "Bi-directional"
SEBL_COMP_NAME	Varchar:75	Siebel Component	S_INT_SEBLCMP (by name)	Name of the Siebel component in the component map that the Siebel field is a part of. Pick-list should only include Siebel components under the same component map.
SEBL_FLD_NAME	Varchar:75	Siebel Field	S_INT_FIELD (by name)	Name of the Siebel field. Pick-list should only include fields from the specified component.
EXT_COMP_NAME	Varchar:75	External Component	S_INT_EXTCMP (by name)	Name of external component in the component map that the external field is a part of. Pick-list should only include external components under the same component map.
EXT_FLD_NAME	Varchar:75	External Field	S_INT_FIELD (by name)	Name of the external field. . Pick-list should only include fields from the specified component.
CALC_EXPR	Varchar:20000	Calculated Expression		Calculated expression value.
PIVOT_KEY_VALUE	Varchar:250	Pivot Key Value		If this field is from a component that is pivot-merged, then this specifies the value of the pivot key.

7 Application Data Model Specification

This section lists additions/changes to the application tables.

7.1 EAI Component Setup Schema

These tables are for internal use only – administration screens do not need to be built on top of them.



7.1.1 S_EAI_APPLVER

This table stores the information about the Siebel and External Applications (and their version) that have an adapter.

Column	**Data Type**	**Foreign Key Table**	**Description**
ROW_ID	Varchar:15		Primary key
NAME$^{(UK,1)}$	Varchar:75		Name of Application
VERSION$^{(UK,2)}$	Varchar:75		Application specific version information. (e.g. for SAP this could be "3.1H" or "4.5B", for PeopleSoft this could be "7.5.2" or "8.0", and for Siebel it could be "5.1" or "6.0")
DESC_TEXT	Varchar:250		Description of the Application

7.1.2 S_EAI_ADAPTER

This table stores information about each adapter.

Column	**Data Type**	**Foreign Key Table**	**Description**
ROW_ID	Varchar:15		Primary key
NAME$^{(UK)}$	Varchar:75		Name of this adapter
DESC_TEXT	Varchar:250		Description for this adapter
SOURCE_FLG	Boolean		True if the adapter can act as a source of messages
SINK_FLG	Boolean		True if the adapter can act as a destination for messages.
SHARED_LIBRARY	Varchar:30		Name of the Shared Library that implements this adapter. Note, the library name should not have a suffix (e.g. ".dll" or ".so").
FACTORY_FUNCTION	Varchar:30		The function to create a new adapter; the prototype is: `Adapter *(const EAISchAdpt *)`
CLASS_NAME	Varchar:30		The class that implements this adapter.

Notes: This schema allows the same SHARED_LIBRARY to be used for multiple adapters, in this case either all the adapters can have the same FACTORY_FUNCTION which will then create the appropriate adapter based on the `EAISchAdpt*` argument, or each adapter can provide a different FACTORY_FUNCTION.

7.1.3 S_EAI_APPL_ADPT

This table stores the information about the Siebel and External Applications that have an adapter.

Column	Data Type	Foreign Key Table	Description
ROW_ID	Varchar:15		Primary key
EAI_APPLVER_ID(UK,1)	Varchar:15	S_EAI_APPL	Key of the Application
EAI_ADAPTER_ID(UK,2)	Varchar:15	S_EAI_ADAPTER	Key of the Adapter

Notes: The reason for a many-to-many relationship between Applications and Adapters is to support cases when a newer version of the Application provides a new adapter (interface) while the previous one is also supported. For example, in Siebel99.1 we provide the EIM Adapter, and in Siebel2000 we provide a new BusComp Adapter but the EIM Adapter is also supported for the Siebel2000 Adapter.

7.1.4 S_EAI_FLOW

This table stores information about the various adapter flows that the EAI Component can execute.

Column	Data Type	Foreign Key Table	Description
ROW_ID	Varchar:15		Primary key
NAME(UK)	Varchar:75		Name of this sequence (e.g. "SAP2SEBL")
DESC_TEXT	Varchar:250		Description of this sequence (e.g. "The SAP to Siebel RFC/IDOC Sequence"
OBJECT_TYPES	Varchar:250		A comma separated list of Integration Objects that the source adapter of this flow is allowed to produce. (e.g. "SAP IDOC: Customer master, SAP IDOC: Master material") . Using the similarly named component parameter can narrow this list; though new object types cannot be added using the component parameter.

7.1.5 S_EAI_ADPT_FLOW

This table stores information about the adapters that make up a flow.

Column	Data Type	Foreign Key Table	Description
ROW_ID	Varchar:15		Primary key
EAI_FLOW_ID(UK,1)	Varchar:15	S_EAI_FLOW	Id of this flow
ELEMENT_NUM(UK,2)	Integer		Position of the element in the sequence
EAI_ADAPTER_ID	Varchar:15	S_EAI_ADAPTER	The adapter for this element

7.2 *Schema for DTE (application tables)*

Administration screens need to be built on top of these tables.

7.2.1 S_EAI_LOOKUPMAP

This is a table for performing lookup mappings between Siebel and External values.

Column	Data Type	Field Name	Foreign Key Table	Usage
LOOKUP_TYPE	Varchar:30	Lookup Map Type	S_LST_OF_ VAL (by name)	Name of the mapping type. There is a LOV type for "EAI Lookup Map Types". Example: "Sebl⇔PSFT Currency Code Mapping"
SEBL_VALUE	Varchar:120	Siebel Value		
EXT_VALUE	Varchar:120	External Value		

7.2.2 S_EAI_DTEOBJ2MP

This table specifies which map(s) should be executed for a given input Integration Object.

Column	Data Type	Field Name	Foreign Key Table	Usage
ROW_ID	Varchar:15			Primary Key
INPUT_INTOBJ _NAME(UK, 1)	Varchar:75	Input Integration Object	S_INTGR_O BJ (by name)	
MAP_NAME(UK,2)	Varchar:75	Integration Object Map	S_INTGR_O BJMAP (by name)	
DIRECTION_C D	Varchar:30			Pick-list: "Inbound to Siebel", "Outbound from Siebel"

7.3 Schema for Peoplesoft Adapter

7.3.1 Staging Tables

They function as temporary store for PSFT adaptor and DTE. DTE operates on tables, but the format of data from an external ERP system can be table, file or message. The use of staging table will standardize the interface to DTE on Siebel side, and let the PSFT adaptor specialize in the format conversion. Except some header columns, a Siebel staging table is modeled to the corresponding PeopleSoft staging table in a one-to-one fashion. This way, PSFT adaptor minimizes its role in data mapping which is already handled by DTE. There are 5 staging tables for PeopleSoft Integrations:

- S_STG_PS_CUSTOMER
- S_STG_PS_PRODUCT
- S_STG_PS_PRICELIST
- S_STG_PS_EMPLOYEE
- S_STG_PS_ORDER

Their column definitions for the application fields are exactly the same as their counter-parts in the PeopleSoft staging tables. But the ***common header columns*** are defined as follows:

Column	Data Type	Foreign Key Table	Description
CIP_MSG_NUM	Number: 15		Same as MSG_NUM in PSFT
CIP_MSG_SEQ_NUM	Number: 15		Same as MSG_SEQ_NUM in PSFT
TRANS_PROGRAM	Varchar: 4	S_LST_OF_VAL	Same as Trans_program in PSFT
TRANSACTION_CODE	Varchar: 4	S_LST_OF_VAL	Same as TRANSACTION_CODE in PSFT
CIP_TXN_FLAG	Varchar: 1	S_LST_OF_VAL	'I' for Siebel Inbound or PSFT Outbound 'O' for Siebel Outbound or PSFT Inbound
CIP_MSG_ACTN	Varchar: 4	S_LST_OF_VAL	Same as CIP_MSG_ACTN (1-Add, 2-Update, 3-Delete, 4-Inactivate, etc)
EAI_INTEG_COMP	Varchar: 15	S_INTEG_COMP	FK to integration comp definition
EAI_BATCH_NUM	Number: 15		Assigned by adaptor to group MSGs that are processed within one polling period, can be mapped to IF_ROW_BATCH_NUM.
EAI_STAT_CODE	Varchar: 5	S_LST_OF_VAL	code to track transaction status
(mapped application fields)			Only application fields in PSFT that are mapped to Siebel

7.3.2 Lookup tables : S_LST_OF_VAL

Type	Name	Val	Description
CIP_MSG_STAT	New	0	
	In process	1	
	Success	2	
	Error	3	
	Picked	4	
	Resubmit	5	
	Ignore	6	
CIP_TXN_ACTION	ADD	1	
	UPDATE	2	
	DELETE	3	
	INACTIVATE	4	
CIP_TXN_FLAG	INBOUND	I	
	OUTBOUND	O	
CIP_TXN_PROG	Customer	001	
	Product	014	
	Employee	015	
	Sales Order	007	
CIP_TXN_CODE	Customer Master	1100	
	Customer Address Sequence	1101	
	Customer Address Master	1102	
	Customer Contact Sequence	1103	
	Customer Contact Master	1104	
	Customer Contact Phone	1105	
	Product Master	1401	
	Product Group	1402	
	Product Group Link	1403	
	Product UOM	1404	
	Product Price	1405	
	Employee Master	0151	
	Employee Employment	0152	
	Employee Job	0153	
	Employee NID	0154	
	Department	0155	

	Location	0156	
	Sales Order Header	0701	
	Sales Order Line	0702	

8 Sub-component Interfaces

8.1 EAI System Fields (for Staging Tables)

There are a number of integration fields that will have a special meaning to the adapters. These fields will have the Siebel System Field flag set to true. Here is a description of the fields:

Field	Data Type	Description
EAI_BATCH_NUM	Number	Batch number of this record. Used to identify groups of records to be retrieved by a given adapter call.
EAI_INT_COMP_ID	Varchar:15	Pointer to integration component definition (S_INT_COMP). Necessary when multiple components share the same table.
EAI_ROW_STAT_CD	Varchar:30	Status: "New", "In Progress", "Success", "Error", "Re-submit"
EAI_ROW_TRANS_CD	Varchar:30	Transaction type: "Create", "Update", or "Delete"
EAI_ROW_GBL_ID	*Varchar:15*	*Reserved for future use. Pointer to global id entry (S_INT_GBLCOMPS) for this row.*
EAI_ROW_SRC_GBL_ID	*Varchar:15*	*Reserved for future use. Pointer to global id entry for the source of this row.*

8.2 Call-Level Error Handling

We will be using the ScfErrorStack facility for returning error status between nested function calls. SMI shell creates the error stack as part of the component initialization for EAI (before CompMain() is called), and provides the macros ERROR_START, ERROR_IFNULL, ERROR_SIGNAL, ..., ERROR_STOP to be used by all modules within that component. The stack of error codes will be automatically translated into appropriate messages and printed in the trace-file corresponding to the EAI component.

There will be an independent facility provided by the adapter interface for providing status information about messages transferred between adapters. This facility would include information about the number of rows accepted and rejected by an adapter. If an adapter encounters a fatal error in its infrastructure (such as a database connection failure), the adapter would return an SCF error. If an adapter rejects some of the data provided by another adapter, it will return an SCF success status along with information about the rows that were rejected.

Example usage of the ScfErrorStack facility in a function is as follows:

```
ERR_START(1, errCode)
    errCode = RunSystem(ruleStr, &retCode, errStr, errStrLen);
    if (errCode != SUCCESS) {
      LogTrace(COMP_TRACE_ALL, "%s %s TELNET %s %d %d %s %s %d", m_pPrintCmd,
            m_pDelCmd, vIp, vport, m_pTimeOut, m_pPath, srvrIp, sport);
      ERR_SIGNAL(1, errCode);
    }

    // check the error returned by resonate command
    if (retCode != SUCCESS) {
      errStr[errStrLen] = '\0';
      sprintf(ruleStr, "%s %s TELNET %s %d %d %s %s %d", m_pPrintCmd,
            m_pDelCmd, vIp, vport, m_pTimeOut, m_pPath, srvrIp, sport);
      if (LIBStrICmp(errStr, "No Such Service") == 0) {
        ERR_SIGNAL4(1, ERR_SCB_SVC_NOTFOUND, sport, vport, srvrIp, vIp);
      }
      else if (LIBStrICmp(errStr, "Error -- Service not found") == 0) {
        ERR_SIGNAL4(1, ERR_SCB_SVC_NOTFOUND, sport, vport, srvrIp, vIp);
      }
      else
        ERR_SIGNAL2(1, ERR_SCB_RESONATE, ruleStr, errStr);
    }

  ERR_END(1);
    ERR_HANDLER(ALL_ERRORS)
    {
      if (bForce != TRUE)
        ERR_PRINT;
    }
  ERR_EXIT(1);
```

Using Integration Objects

Exhibit C

Using Integration Objects 2

About This Chapter

This chapter describes how to create and use Siebel EAI integration objects to accomplish your integration requirements.

Understanding the Terminology

This chapter deals with concepts that are often referred to using different terminology from one system to another. This section has been included to clarify the information in this chapter by providing a standard terminology for these concepts.

Table 2-1. Terminology

Term	Description
metadata	Data that describes data. For example, the term datatype describes data elements such as char, int, Boolean, time, date, and float.
Siebel business object	A Siebel object type that creates a logical business model using links to tie together a set of interrelated business components. The links provide the one-to-many relationships that govern how the business components interrelate in this business object.
component	A constituent part of any generic object.
Siebel business component	A Siebel object type that defines a logical representation of columns in one or more database tables. A business component collects columns from the business component's base table, its extension tables, and its joined tables into a single structure. Business components provide a layer of abstraction over tables. Applets in Siebel applications reference business components; they do not directly reference the underlying tables.
field	A generic reference to a data structure that can contain one data element.
Siebel integration component field	A data structure that can contain one data element in a Siebel integration component.
Siebel integration component	A constituent part of a Siebel integration object.
integration object	Refers to an integration object of any type, including the Siebel integration object, the SAP BAPI integration object, and the SAP IDOC integration objects.
integration object instance	Actual data, usually the result of a query or other operation, which is passed from one business service to another, structurally modeled on a Siebel integration object.
Siebel integration object	An object stored in the Siebel repository that represents some Siebel business object.
integration message	A bundle of data consisting of two major parts: header information that describes what should be done with or to the message itself; and instances of integration objects, that is, data in the structure of the integration object.

Integration Object vs. Integration Object Instance

Understanding the difference between integration objects and integration object instances is important, especially in regard to the way they are discussed in this chapter.

An integration object, in the context of EAI, is metadata; that is, it is a generalized representation or model of a particular set of data. It is a schema of a particular thing.

An integration object instance, on the other hand, is actual data organized in the format or structure of the integration object. Consider a very simple example, using partial data, in Figure 2-1.

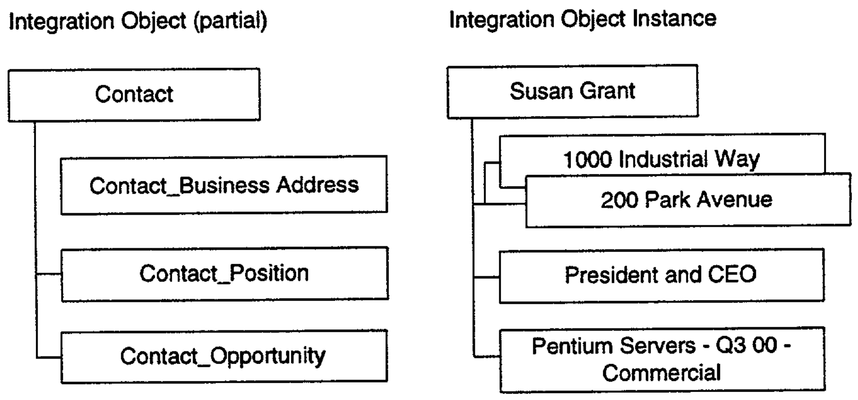


Figure 2-1. Integration Object versus Integration Object Instance

Any discussion of integration objects will include clarifying terms—for example, metadata or Siebel—to help make the distinction for you.

Understanding Siebel Integration Objects

Siebel integration objects allow you to represent Siebel business objects, SAP IDOCs, and SAP BAPIs as common structures that the EAI infrastructure can understand. Integration objects adhering to a set of structural conventions can be traversed and transformed programmatically, using Siebel eScript objects, methods, and functions, described in Chapter 5, "Transforming Integration Data."

The typical integration project involves transporting data from one application to another. You might, for example, want to synchronize data from a back-office system with the data in your Siebel application. You might want to be able to generate a quote in the Siebel application and perform a query against your ERP system transparently. In the context of Siebel EAI, data is transported in the form of an *integration message*. A message, in this context, typically consists of header data that identifies the message type and structure, and a body that contains one or more instances of data—for example, orders, accounts, or employee records.

When planning your integration project, you should consider several issues:

- How much data transformation will your message require?
- At what point in the process will you perform the data transformation?
- Will you require a confirmation message response to the sender?
- Are there data items in the originating data source that should not be replicated in the receiving data source, or that should replace existing data in the receiving data source?

These are just some of the design issues you need to consider when planning your integration project. A solid grasp of both the Siebel business object architecture and the format of your external data is essential to accomplishing your goals.

This guide can help you understand how Siebel EAI represents the Siebel business object structure. Additionally, it provides descriptions of how Siebel EAI represents external SAP R/3 structures.

The Structure of Siebel Integration Objects

The Siebel integration object provides a structure that accommodates many types of data. Most specifically, pre-built EAI integration objects describe the structure of Siebel business objects, SAP IDOCs, and SAP BAPIs. Most integration projects will require the use of an integration object that describes Siebel business objects, either in an outbound direction—a *query* operation against a Siebel integration object—or an inbound direction—a *synchronize* operation against a Siebel integration object.

This chapter continues with descriptions of how to create integration objects. The initial process—using the Integration Object Builder—is essentially the same for all types of integration objects currently supported.

Caution: You should avoid using or making any modifications to integration objects in the EAI Design project. Using or modifying any objects in the EAI Design project can result in unexpected and unpredictable results.

Creating Siebel Integration Objects

You use the Integration Object Builder in Siebel Tools to create new Siebel integration objects. This wizard walks you through the process of selecting objects, either from the Siebel repository or from an external system, on which you can base your new Siebel integration object. The Integration Object Builder builds a list of valid components from which you can choose the components to include in your Siebel integration object. Keep in mind that the Siebel integration object represents the structure—that is, the *metadata* representation—of an object.

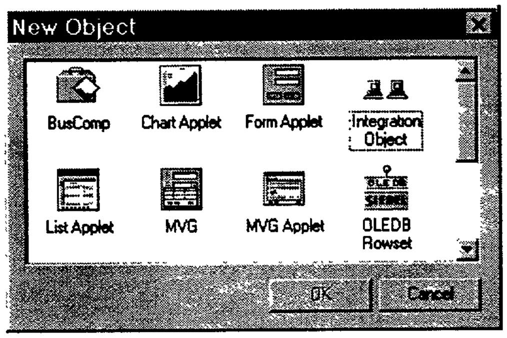
Caution: The Integration Object Builder provides a partial rendering of your data in the integration object format. You must review the integration object definition and complete the definition of your requirements. In particular, you should confirm that user key definitions or other identifiers are defined properly. You might need to enter user keys manually or inactivate unused keys in Siebel Tools. You should not expect to use the integration object without modification.

To create a new integration object

1 Start Siebel Tools.

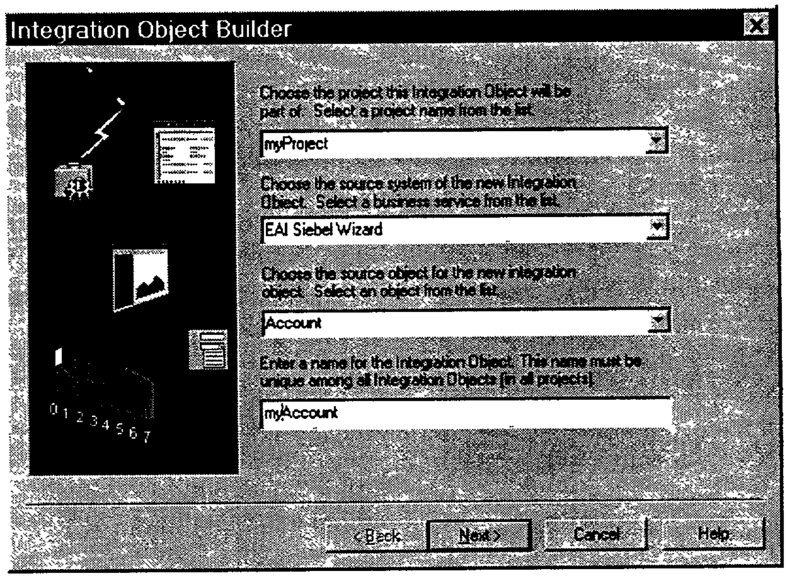
2 Create a new project and lock it, or lock an existing project.

3 Choose File → New Object... to display the New Object dialog box.



4 Select the Integration Object icon and click OK.

The first panel of the Integration Object Builder wizard appears.



Choose the project this Integration Object will be part of. Select a project name from the list.

myProject

Choose the source system of the new Integration Object. Select a business service from the list.

EAI Siebel Wizard

Choose the source object for the new integration object. Select an object from the list.

Account

Enter a name for the Integration Object. This name must be unique among all Integration Objects (in all projects).

myAccount

5 Fill in the four fields on the dialog box:

- **a** Choose a project in which your new Integration Object will exist. The project must be locked.
- **b** Choose a business service that interfaces with the source system, whether that is the Siebel application or an external application.
- **c** Choose the source object. This is the object that will be the model for the new Siebel integration object.
- **d** Type a unique name in the field for the new Siebel integration object.

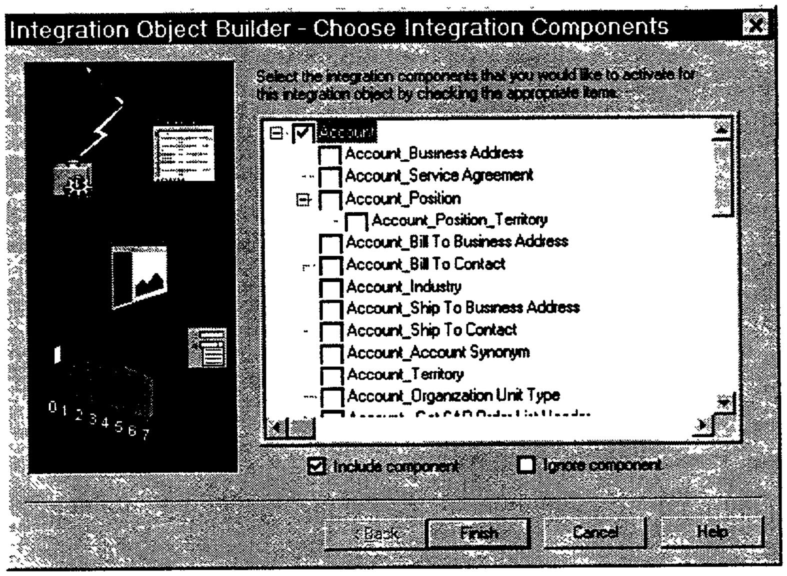
  The name of an integration object must be unique among all other integration objects.

6 Click Next.

The next wizard displays the available components of the object you chose.

7 Click on any of the boxes next to the components listed here, to ignore that component.

If you do not include a component, you will be unable to transport data for that component between the Siebel eBusiness Application and another system.


Integration Object Builder - Choose Integration Components

Select the integration components that you would like to activate for this integration object by checking the appropriate items.

```
⊟ [✓] Account
      [ ] Account_Business Address
      [ ] Account_Service Agreement
    ⊟ [ ] Account_Position
        [ ] Account_Position_Territory
      [ ] Account_Bill To Business Address
      [ ] Account_Bill To Contact
      [ ] Account_Industry
      [ ] Account_Ship To Business Address
      [ ] Account_Ship To Contact
      [ ] Account_Account Synonym
      [ ] Account_Territory
      [ ] Account_Organization Unit Type
```

☑ Include component ☐ Ignore component

Note the following rule for selecting subcomponents.

Any component that has a plus sign (+) next to it is a parent in a parent-child relationship with one or more child components. If you deselect the parent component, all children below that component are deselected as well. You cannot include a child component without also including the parent. The Integration Object Builder enforces this rule by automatically selecting the parent of any child you choose to include.

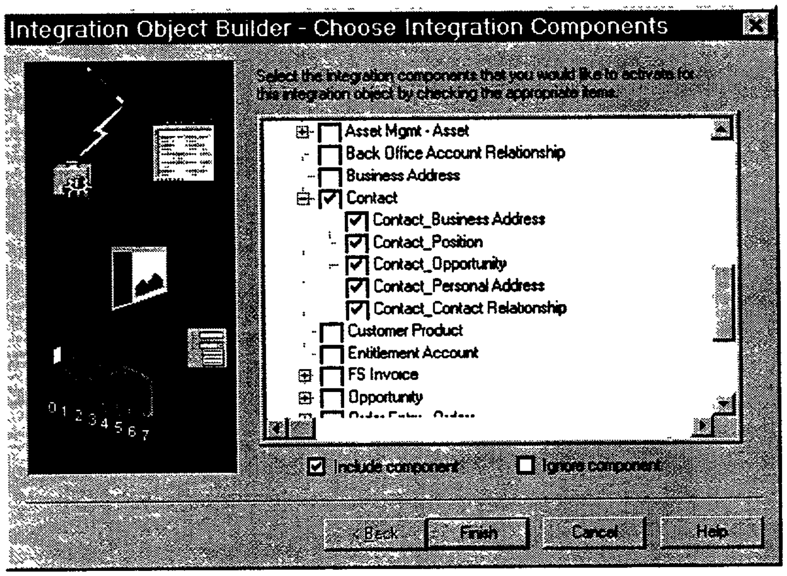
For example, assume you had chosen to build your Siebel integration object on the Siebel Account business object.

8 Deselect the Account integration component at the top of the scrolling list.

This action deselects the entire tree below Account.

9 Select the Contact component.

None of the components below Contact are selected. You must select the ones you want individually. The Account component above Contact is selected, however, because it is the parent of Contact.

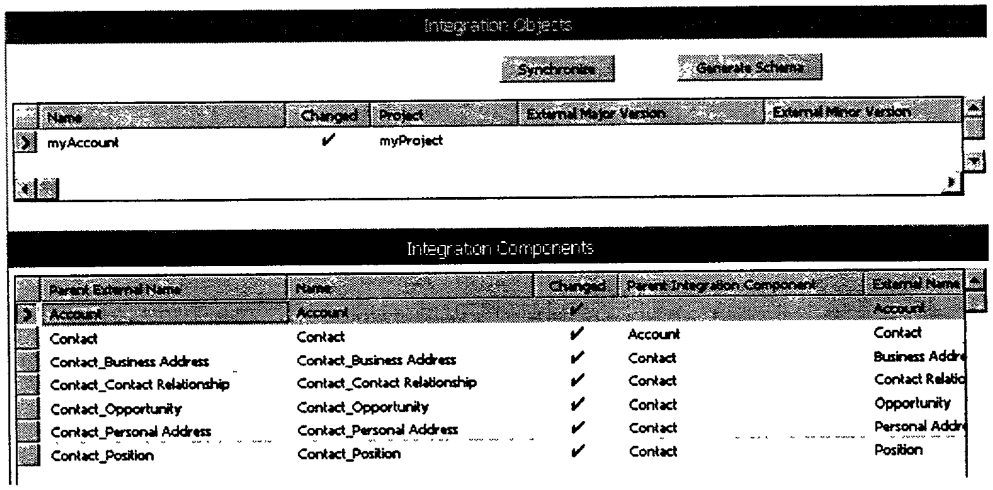

Integration Object Builder - Choose Integration Components

Select the integration components that you would like to activate for this integration object by checking the appropriate items.

- [ ] Asset Mgmt - Asset
- [ ] Back Office Account Relationship
- [ ] Business Address
- [✓] Contact
  - [✓] Contact_Business Address
  - [✓] Contact_Position
  - [✓] Contact_Opportunity
  - [✓] Contact_Personal Address
  - [✓] Contact_Contact Relationship
- [ ] Customer Product
- [ ] Entitlement Account
- [ ] FS Invoice
- [ ] Opportunity

[✓] Include component [ ] Ignore component

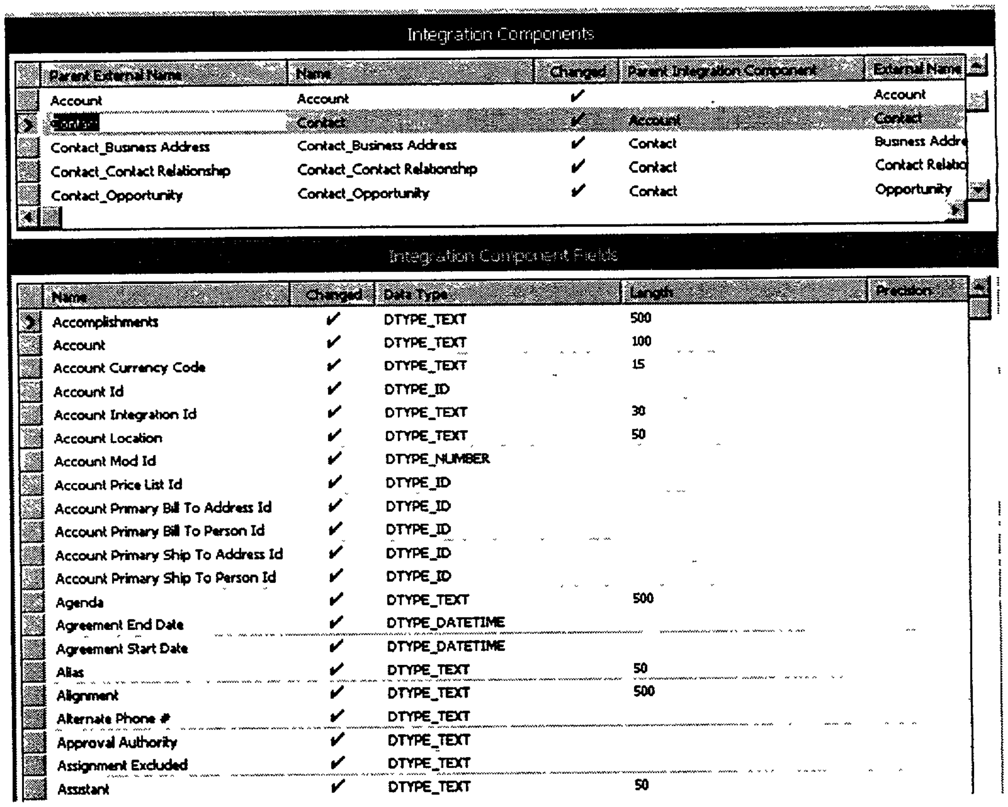
10 Click Finish to complete the process of creating a new Siebel integration object. Your new Siebel integration object appears in the list of integration objects in Siebel Tools.



Name	Changed	Project	External Major Version	External Minor Version
myAccount	✓	myProject		

Parent External Name	Name	Changed	Parent Integration Component	External Name
Account	Account	✓		Account
Contact	Contact	✓	Account	Contact
Contact_Business Address	Contact_Business Address	✓	Contact	Business Addr
Contact_Contact Relationship	Contact_Contact Relationship	✓	Contact	Contact Relatio
Contact_Opportunity	Contact_Opportunity	✓	Contact	Opportunity
Contact_Personal Address	Contact_Personal Address	✓	Contact	Personal Addr
Contact_Position	Contact_Position	✓	Contact	Position

On the Integration Components screen, the Account integration component is the only component that has a blank field in the Primary Integration Component column. The blank field identifies Account as the root component. The Siebel integration object also contains the other components selected, such as Contact and its child components.



Parent External Name	Name	Changed	Parent Integration Component	External Name
Account	Account	✓		Account
Contact	Contact	✓	Account	Contact
Contact_Business Address	Contact_Business Address	✓	Contact	Business Addr
Contact_Contact Relationship	Contact_Contact Relationship	✓	Contact	Contact Relatio
Contact_Opportunity	Contact_Opportunity	✓	Contact	Opportunity

Name	Changed	Data Type	Length	Precision
Accomplishments	✓	DTYPE_TEXT	500	
Account	✓	DTYPE_TEXT	100	
Account Currency Code	✓	DTYPE_TEXT	15	
Account Id	✓	DTYPE_ID		
Account Integration Id	✓	DTYPE_TEXT	30	
Account Location	✓	DTYPE_TEXT	50	
Account Mod Id	✓	DTYPE_NUMBER		
Account Price List Id	✓	DTYPE_ID		
Account Primary Bill To Address Id	✓	DTYPE_ID		
Account Primary Bill To Person Id	✓	DTYPE_ID		
Account Primary Ship To Address Id	✓	DTYPE_ID		
Account Primary Ship To Person Id	✓	DTYPE_ID		
Agenda	✓	DTYPE_TEXT	500	
Agreement End Date	✓	DTYPE_DATETIME		
Agreement Start Date	✓	DTYPE_DATETIME		
Alias	✓	DTYPE_TEXT	50	
Alignment	✓	DTYPE_TEXT	500	
Alternate Phone #	✓	DTYPE_TEXT		
Approval Authority	✓	DTYPE_TEXT		
Assignment Excluded	✓	DTYPE_TEXT		
Assistant	✓	DTYPE_TEXT	50	

To view the fields that comprise each integration component, select a component from the integration component list in Siebel Tools. The list of fields for that component is displayed in the Integration Component Fields applet.

Synchronizing Integration Objects

Business objects often require updates to their definitions to account for changes in data type, length, edit format, or other properties. Alteration of database metadata is not uncommon, yet it can cause undesirable effects on your integration projects if you do not update your integration objects to account for these updates.

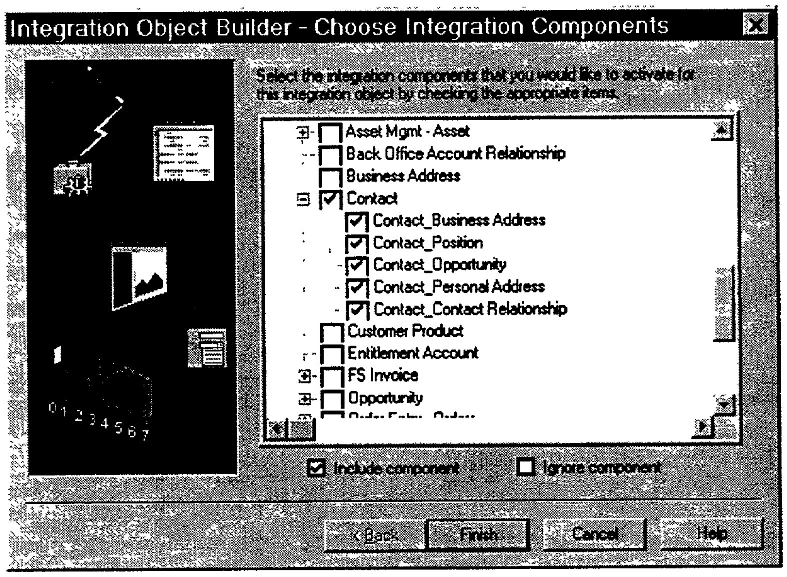
Synchronization Considerations

To help simplify the synchronization task, Siebel EAI provides a synchronization utility. The procedure to synchronize your integration object and its underlying business object is straightforward. Nonetheless, you should spend time reviewing modifications you have made to your integration objects to make sure that you do not inadvertently alter them by performing a synchronization.

Synchronization Rules

During the synchronization process, the Synchronization wizard follows particular update rules. Consider a simple example, involving the Siebel Account integration object with only Contact and its child components marked as active in the object.

To visualize this example, take a look at Figure 2-2.



Integration Object Builder - Choose Integration Components

Select the integration components that you would like to activate for this integration object by checking the appropriate items.

- [ ] Asset Mgmt - Asset
- [ ] Back Office Account Relationship
- [ ] Business Address
- [✓] Contact
  - [✓] Contact_Business Address
  - [✓] Contact_Position
  - [✓] Contact_Opportunity
  - [✓] Contact_Personal Address
  - [✓] Contact_Contact Relationship
- [ ] Customer Product
- [ ] Entitlement Account
- [ ] FS Invoice
- [ ] Opportunity

Figure 2-2. Example of Selected Integration Components

Of course, because the Account component is the parent of Contact, it also is selected, even though you cannot see it in Figure 2-2.

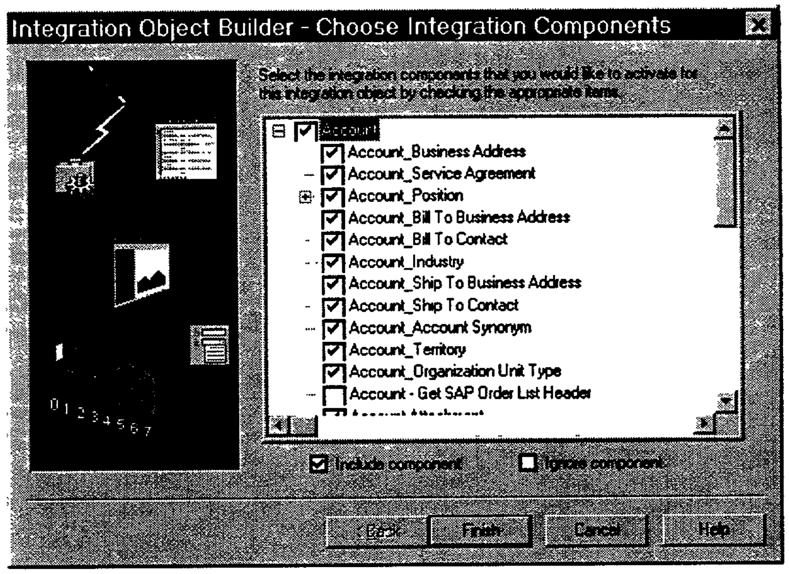
First, take a look at the steps required to synchronize an entire object and all of its components.

To start the Synchronization wizard

1 In Siebel Tools, on the Integration Objects screen, click the Siebel integration object you want to synchronize.

NOTE: If you are using the SAP BAPI or SAP IDOC integration objects, you also follow this general procedure to perform a synchronization operation.

The process of retrieving Siebel integration object and Siebel business object definitions can take varying amounts of time depending on the size and detail of the selected objects.



Select the integration components that you would like to activate for this integration object by checking the appropriate items.

- [✓] Account
  - [✓] Account_Business Address
  - [✓] Account_Service Agreement
  - [✓] Account_Position
  - [✓] Account_Bill To Business Address
  - [✓] Account_Bill To Contact
  - [✓] Account_Industry
  - [✓] Account_Ship To Business Address
  - [✓] Account_Ship To Contact
  - [✓] Account_Account Synonym
  - [✓] Account_Territory
  - [✓] Account_Organization Unit Type
  - [ ] Account - Get SAP Order List Header

☑ Include component ☐ Ignore component

2 Check all of the boxes beside the objects and components you want to include in the synchronization of your Siebel integration object.

The process of performing the synchronization can take some time, depending on the complexity of the selected objects.

3 Click Finish to synchronize the Siebel integration object and the Siebel business object.

When you synchronize the Siebel integration object and the Siebel business object, the Synchronization wizard performs update, insert, and delete operations to make the Siebel integration object look like the representation of the Siebel business object you chose, by selecting or deselecting components.

The general cases of how the wizard updates the Siebel integration object include:

- Completely updating the object and its components
- Updating some components and deleting others

Updating the Entire Integration Object

After initiating the Synchronization wizard, if you leave all the boxes checked in the wizard, the following condition applies:

The wizard always creates a new integration object in memory. If the Siebel business object upon which it's based has been changed, then the new, in-memory integration object will be different than the integration object that exists in the database. In that case, the wizard will synchronize the original, out-of-date integration object with the new, in-memory integration object.

Figure 2-3 illustrates this concept.

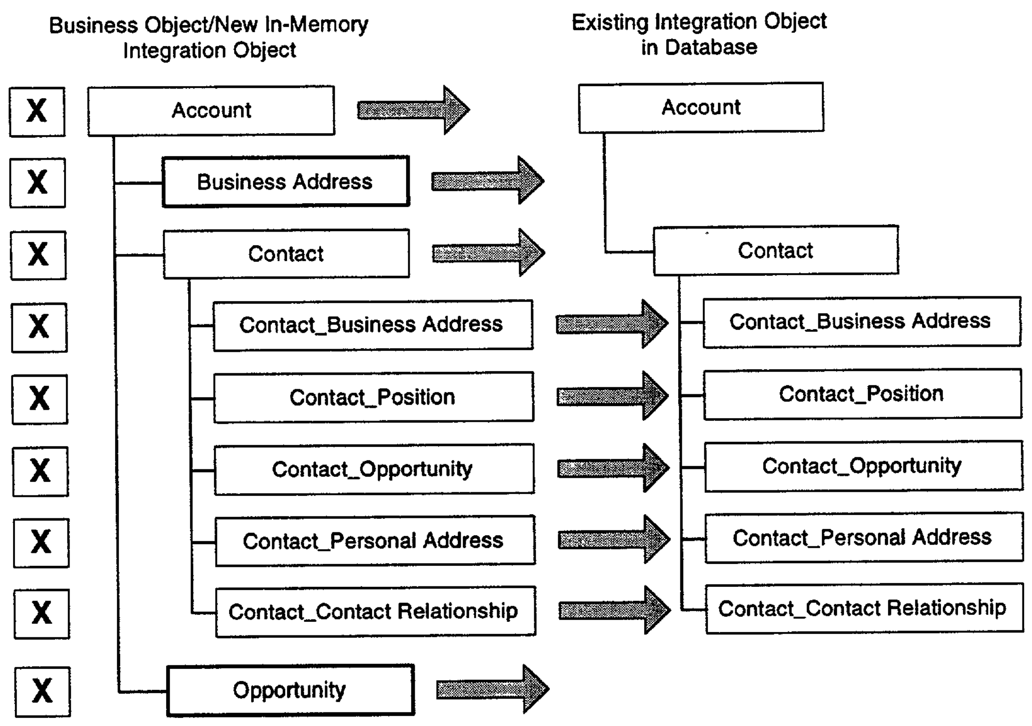


Figure 2-3. Synchronizing the Complete Integration Object

If you look at Figure 2-4, you can see how the resulting integration object will be structured after the synchronization.

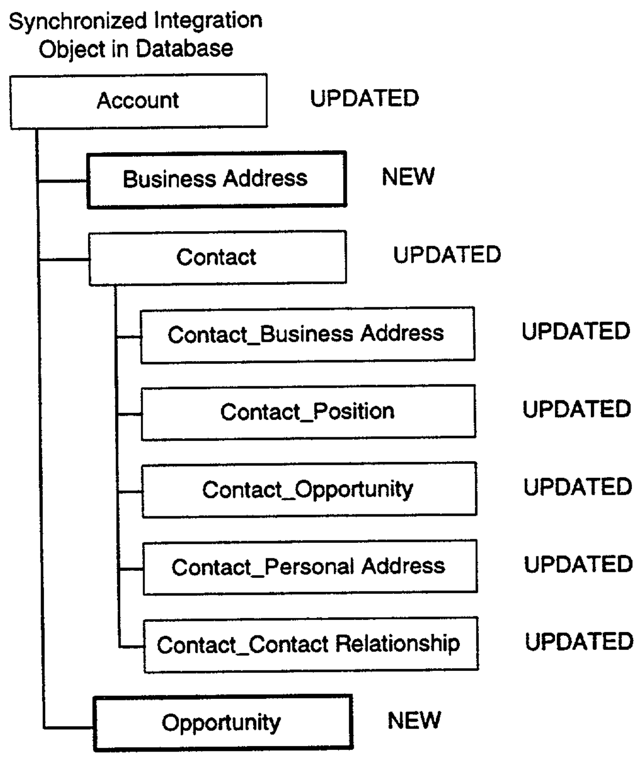


Figure 2-4. Completely Updated Integration Object

The integration object now contains two new components, *Business Address* and *Opportunity*. All other components have been updated with the definitions of the corresponding components in the business object.

Deleting a Component from the Integration Object

If you choose to deselect a component on the Synchronization wizard, you specify to the wizard that it should delete any component in the integration object that matches this criterion.

The integration object that exists in the database has a component with the same External Name, External Name Sequence, and External Name Context as the deselected component in the new, in-memory integration object.

Figure 2-5 illustrates this concept.

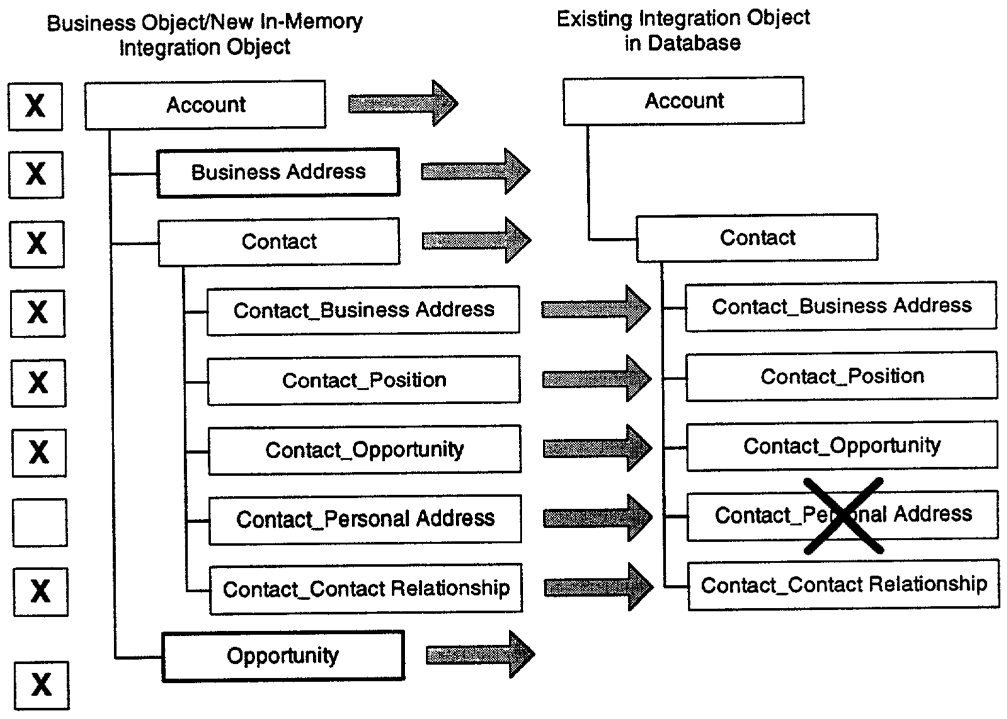


Figure 2-5. Deleting a Component from the Integration Object

This example is intended to show how you might experience unexpected results by deselecting components. On the other hand, if you actually want to delete a particular component from the integration object, this method will accomplish your goal. Figure 2-6 shows the integration object after synchronization.

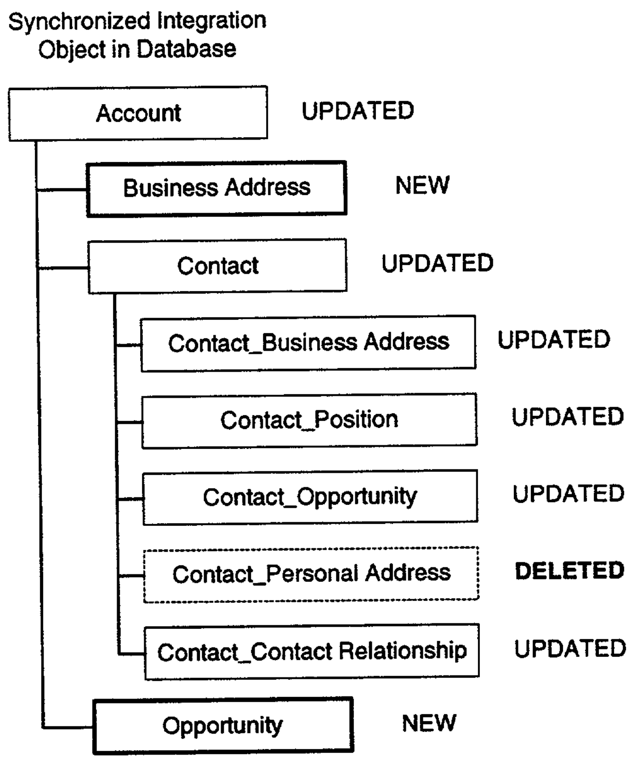


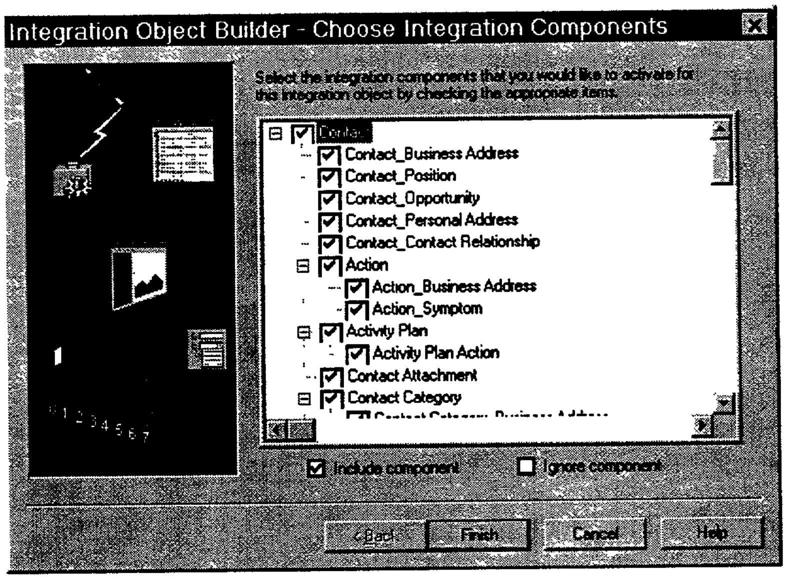
Figure 2-6. Synchronization Resulting in a Deleted Component

The component *Contact_Personal Address* has been deleted. When you use this integration object, you will be unable to pass data for that component between the Siebel application and an external application.

Using the EAI Siebel Wizard

You can create integration objects that represent Siebel business objects by using the EAI Siebel Wizard. During the process of creating a new integration object, described in "Creating Siebel Integration Objects" on page 2-7, you can choose the EAI Siebel Wizard as the business service to help create the object. This wizard understands the structure of Siebel business objects. The wizard returns a list of the available business objects from which you can choose one on which to base your integration object, as shown in Figure 2-7.

The wizard also returns a list of the available components contained within the object you chose. When you select certain components in the wizard, you are activating those components in your integration object. Your integration object actually contains the entire structural definition of the business object you selected on the first wizard panel. Only the components you checked, or left selected, are active within your integration object. That means that any instances you retrieve of that integration object will contain only data represented by the selected components.


Integration Object Builder - Choose Integration Components

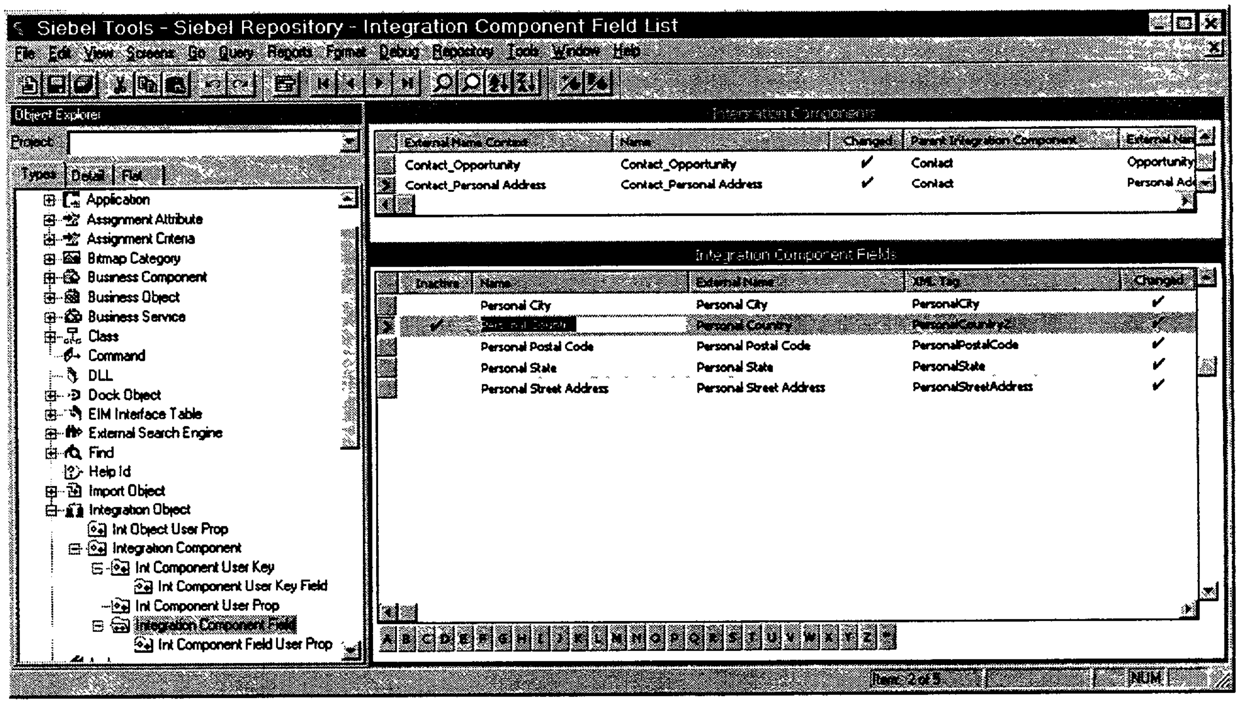
Select the integration components that you would like to activate for this integration object by checking the appropriate items.

- ✓ Contact
  - ✓ Contact_Business Address
  - ✓ Contact_Position
  - ✓ Contact_Opportunity
  - ✓ Contact_Personal Address
  - ✓ Contact_Contact Relationship
  - ✓ Action
    - ✓ Action_Business Address
    - ✓ Action_Symptom
  - ✓ Activity Plan
    - ✓ Activity Plan Action
  - ✓ Contact Attachment
  - ✓ Contact Category

[✓] Include component [ ] Ignore component

Figure 2-7. Activated Components in the Contact Integration Object

After the wizard creates your integration object, you can edit the object in Siebel Tools, as shown in Figure 2-8. You might choose to drill down into the integration components and activate or inactivate particular components or even particular fields within one or more components.



External Name Context	Name	Changed	Parent Integration Component	External Name
Contact_Opportunity	Contact_Opportunity	✓	Contact	Opportunity
Contact_Personal Address	Contact_Personal Address	✓	Contact	Personal Add

Inactive	Name	External Name	XML Tag	Changed
	Personal City	Personal City	PersonalCity	✓
✓	Personal Country	Personal Country	PersonalCountry2	✓
	Personal Postal Code	Personal Postal Code	PersonalPostalCode	✓
	Personal State	Personal State	PersonalState	✓
	Personal Street Address	Personal Street Address	PersonalStreetAddress	✓

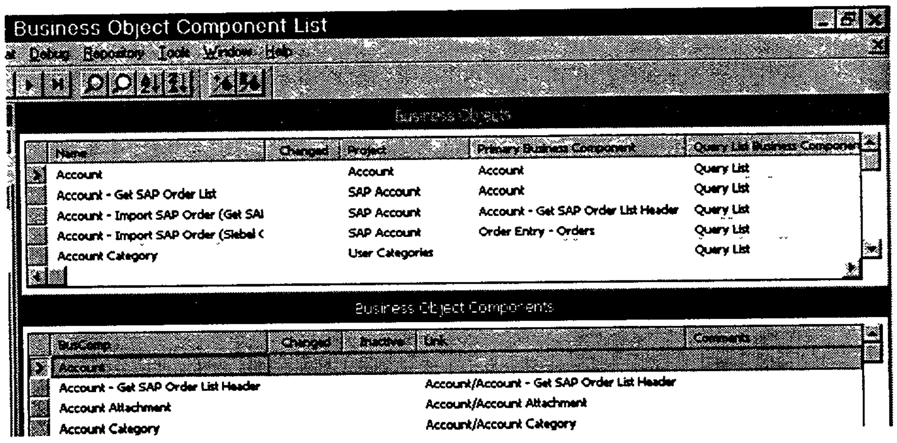
Figure 2-8. Inactivating an Integration Component Field

Understanding the Structure of Siebel Integration Objects

Siebel business objects conform to a particular structure in memory. Although it is generally not necessary to consider this structure when working with Siebel applications, when you are planning and designing an integration project it is helpful to understand how an EAI integration object represents that internal structure.

An integration object consists of one Parent Integration Component, sometimes referred to as the root component or the primary integration component. The Parent Integration Component corresponds to the primary business component of the business object you chose as the model for your integration object.

Take a look at the Account business object in Siebel Tools, as shown in Figure 2-9.


Business Object Component List
Debug Repository Tools Window Help
Business Objects
Business Object Components

Name	Changed	Project	Primary Business Component	Query List Business Component
Account		Account	Account	Query List
Account - Get SAP Order List		SAP Account	Account	Query List
Account - Import SAP Order (Get SAI		SAP Account	Account - Get SAP Order List Header	Query List
Account - Import SAP Order (Siebel (		SAP Account	Order Entry - Orders	Query List
Account Category		User Categories		Query List

BusComp	Changed	Inactive	Link	Comments
Account				
Account - Get SAP Order List Header			Account/Account - Get SAP Order List Header	
Account Attachment			Account/Account Attachment	
Account Category			Account/Account Category	

Figure 2-9. Account Parent Business Component

For example, assume you chose the Account business object (on the first panel of the Integration Object Builder) on which to base your integration object *myAccount_01*. The Account business object in Siebel Tools has an Account business component as its primary business component. In the *myAccount_01* integration object, all of the child components will be represented as children of the primary business component named Account.

Each child component can have one or more child components. In Siebel Tools, if you look at the integration components for an integration object you have created, you will see that each component can have one or more fields. Figure 2-10 on page 2-24 illustrates a partial view of a Siebel integration object based on the Account business object, with the Business Address component and the Contact component activated.

The structure shown in Figure 2-10 represents part of an integration object. The Account parent integration component can have both fields *and* child integration components. Each integration component can also have child integration components and fields. A structure of this sort represents the *metadata* of an Account integration object. You might choose to inactivate components and fields. In that way, you can define the structure, which, in turn, will shape the integration object instances entering or leaving the system.

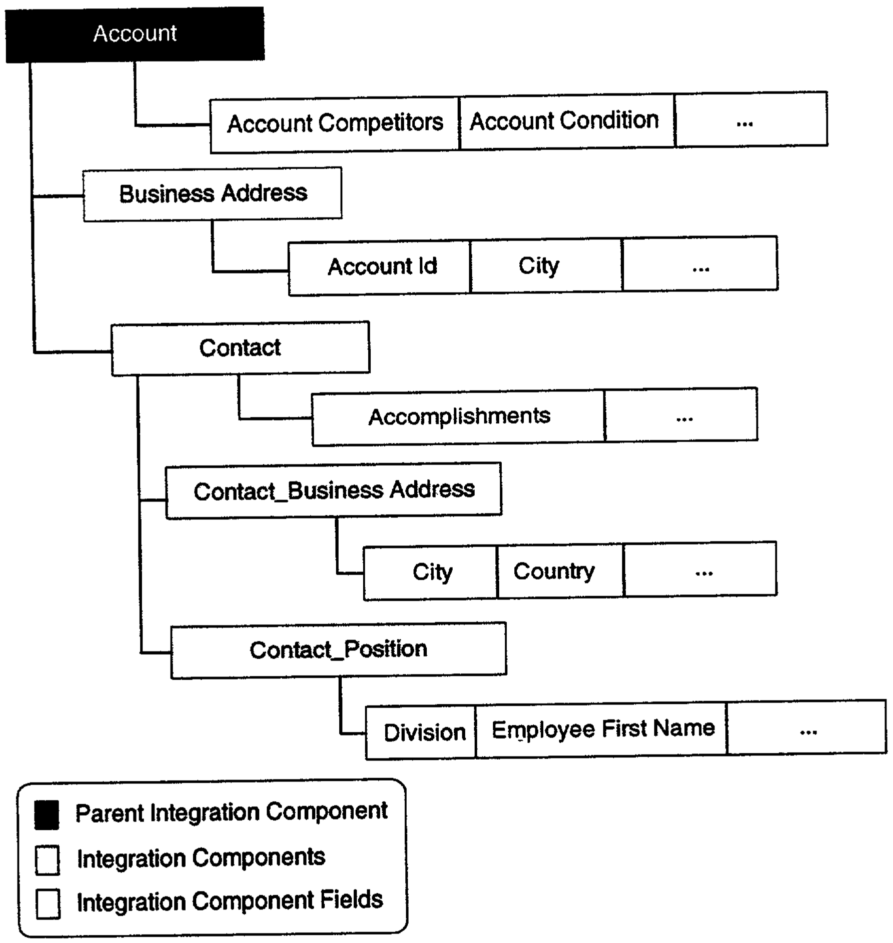


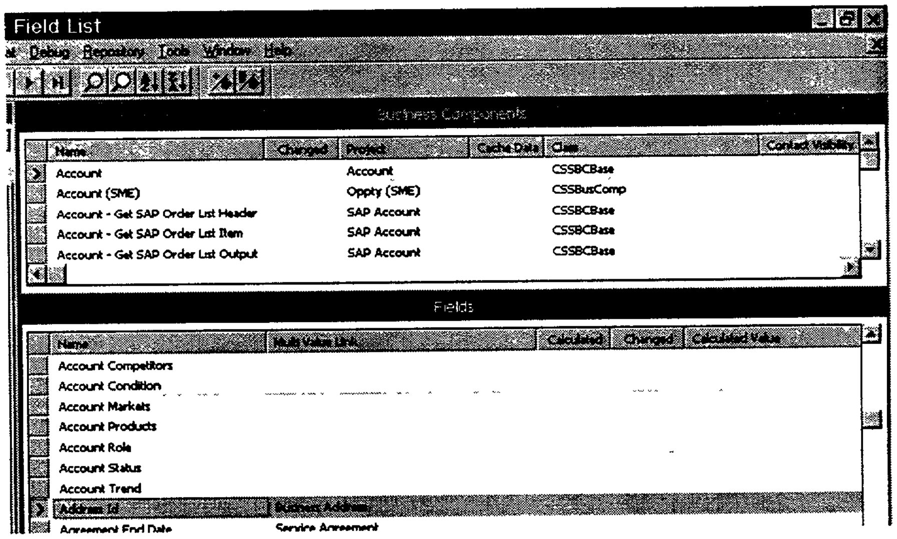
Figure 2-10. Representation of Partial Account Integration Object

Representing Multi-Value Groups

Multi-value groups (MVGs) are used within Siebel business components to represent database multi-valued attributes. MVGs can be one of two types: regular MVGs and MVG Associations.

In Siebel Tools, if a Siebel business component contains an MVG, the MVG will be represented in several screens.

The Siebel business component—for example, Account—contains a field that is an MVG, which appears in the Fields List on the Business Components screen. For example, in Figure 2-11, the Address Id field is an MVG in the Account business component.

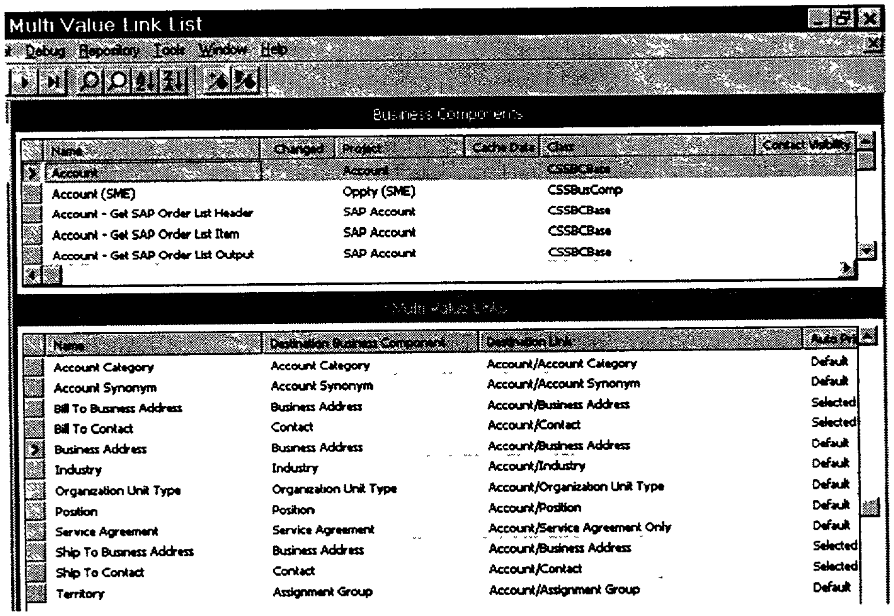


Name	Changed	Project	Cache Data	Class	Contact Visibility
Account		Account		CSSBCBase	
Account (SME)		Oppty (SME)		CSSBusComp	
Account - Get SAP Order List Header		SAP Account		CSSBCBase	
Account - Get SAP Order List Item		SAP Account		CSSBCBase	
Account - Get SAP Order List Output		SAP Account		CSSBCBase	

Name	Multi Value Link	Calculated	Changed	Calculated Value
Account Competitors				
Account Condition				
Account Markets				
Account Products				
Account Role				
Account Status				
Account Trend				
Address Id	Business Address			
Agreement End Date	Service Agreement			

Figure 2-11. Address Id MVG Field in the Account Business Component

The Multi Value Link field, in this example, has the value Business Address. If you go to the Multi Value Link screen, shown in Figure 2-12, you can see that the Business Address multi value link has, as its Destination Business Component, the value Business Address.

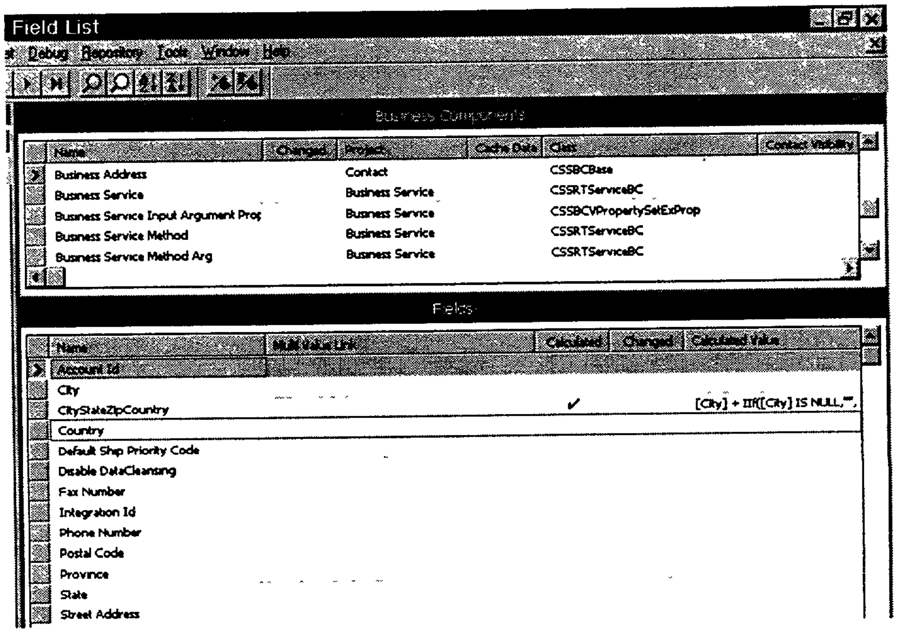


Name	Changed	Project	Cache Data	Class	Contact Visibility
Account		Account		CSSBCBase	
Account (SME)		Oppty (SME)		CSSBusComp	
Account - Get SAP Order List Header		SAP Account		CSSBCBase	
Account - Get SAP Order List Item		SAP Account		CSSBCBase	
Account - Get SAP Order List Output		SAP Account		CSSBCBase	

Name	Destination Business Component	Destination Link	Auto Pri
Account Category	Account Category	Account/Account Category	Default
Account Synonym	Account Synonym	Account/Account Synonym	Default
Bill To Business Address	Business Address	Account/Business Address	Selected
Bill To Contact	Contact	Account/Contact	Selected
Business Address	Business Address	Account/Business Address	Default
Industry	Industry	Account/Industry	Default
Organization Unit Type	Organization Unit Type	Account/Organization Unit Type	Default
Position	Position	Account/Position	Default
Service Agreement	Service Agreement	Account/Service Agreement Only	Default
Ship To Business Address	Business Address	Account/Business Address	Selected
Ship To Contact	Contact	Account/Contact	Selected
Territory	Assignment Group	Account/Assignment Group	Default

Figure 2-12. Destination Business Component

This means that another business component named Business Address that contains the fields that are collectively represented by Address Id in the Account business component, as shown in Figure 2-13.



Name	Changed	Project	Cache Data	Class	Contact Visibility
Business Address		Contact		CSSBCBase	
Business Service		Business Service		CSSRTServiceBC	
Business Service Input Argument Prop		Business Service		CSSBCVPropertySetExProp	
Business Service Method		Business Service		CSSRTServiceBC	
Business Service Method Arg		Business Service		CSSRTServiceBC	

Name	Multi Value Link	Calculated	Changed	Calculated Value
Account Id				
City				
CityStateZipCountry		✓		[City] + IIf([City] IS NULL,"",
Country				
Default Ship Priority Code				
Disable DataCleansing				
Fax Number				
Integration Id				
Phone Number				
Postal Code				
Province				
State				
Street Address				

Figure 2-13. Business Address Business Component

A more graphical way to represent this relationship might be something like that shown in Figure 2-14.

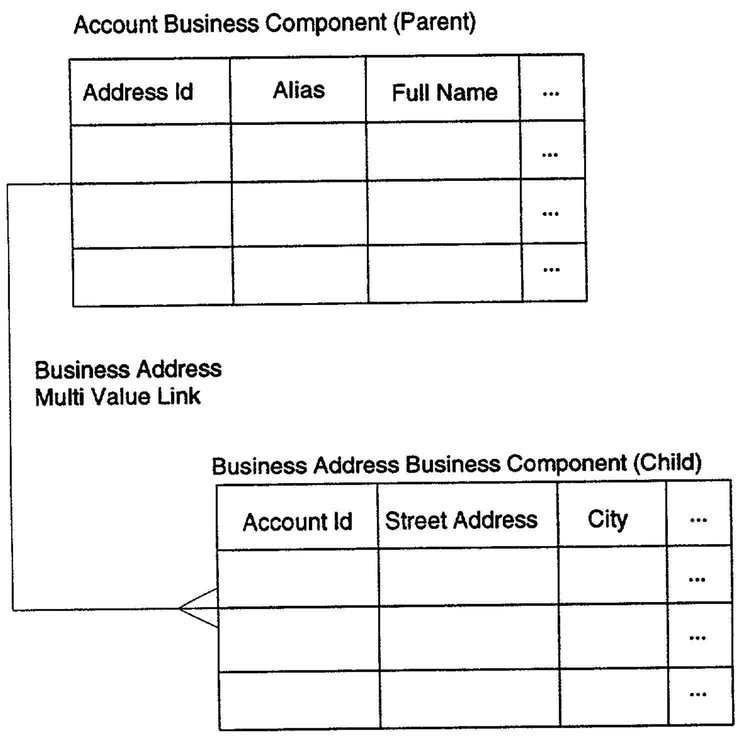


Figure 2-14. Address Id Field and Business Address MVG

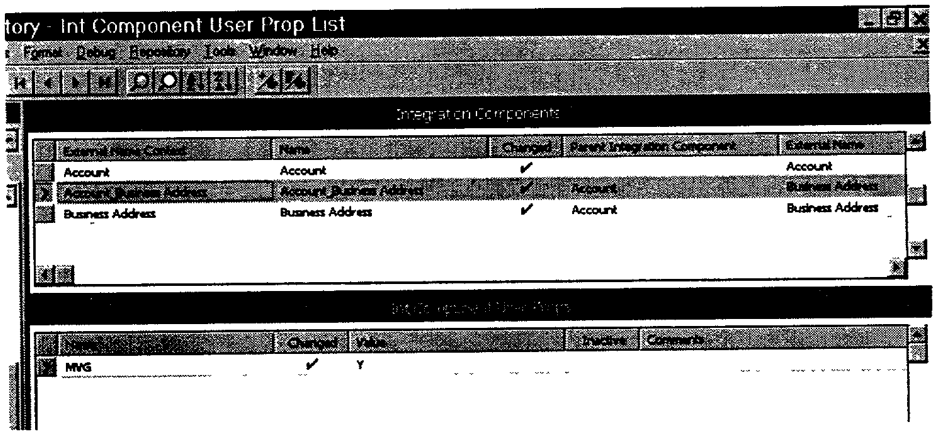
This more table-like representation shows how the Business Address multi value link connects the two business components. The Address Id value points to the Business Address business component, which contains the multiple fields that make up the MVG.

It is important to note that two business components are used to represent an MVG, as in this example.

To create a Siebel integration component to represent this construct, it is necessary also to create two integration components:

- The first integration component represents the parent business component. In the example, this is the Account business component. This integration component contains only the fields that are defined in the parent business component, but which are not based on MVGs. The Multi Value Link column and the Multi-Valued column contain `null` values for these fields.
- The second integration component represents the MVG business component. In the example, this is the Business Address business component. The second integration component has one integration field for each field based on the given MVG in the parent business component. An integration component user property will be set on this integration component to tell the EAI Siebel Adapter that it is based on an MVG business component. If the MVG is a regular MVG, the user property is named *MVG*. If the MVG is an Association MVG, then the user property is named *MVGAssociation*. In both cases, the value of the user property is *Y*.

Figure 2-15 shows an integration component based on an MVG and its user property value in Siebel Tools.

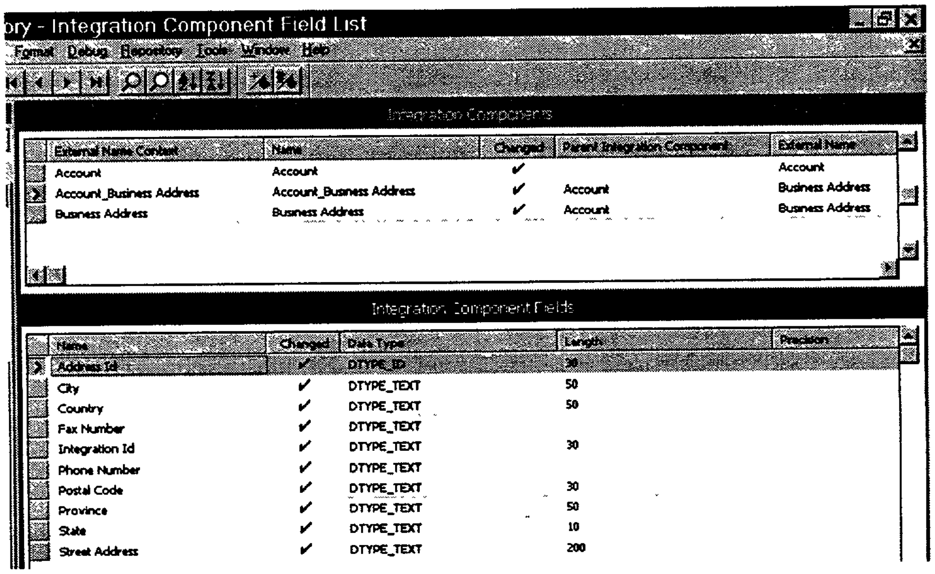

tory - Int Component User Prop List
Format Debug Repository Tools Window Help
Integration Components

External Name Context	Name	Changed	Parent Integration Component	External Name
Account	Account	✓		Account
Account Business Address	Account Business Address	✓	Account	Business Address
Business Address	Business Address	✓	Account	Business Address

Name	Changed	Value	Inactive	Comments
MVG	✓	Y		

Figure 2-15. Integration Component Based on MVG Business Component

The EAI Siebel Adapter needs to know the names of the MVG fields as they are known in the parent business component—in this example, Account—and also the names of the MVG fields as they are known in the business component that represents the MVG—in our example, Business Address. The integration component fields, as shown in Figure 2-16, represent the MVG.


ory - Integration Component Field List
Format Debug Repository Tools Window Help
Integration Components
Integration Component Fields

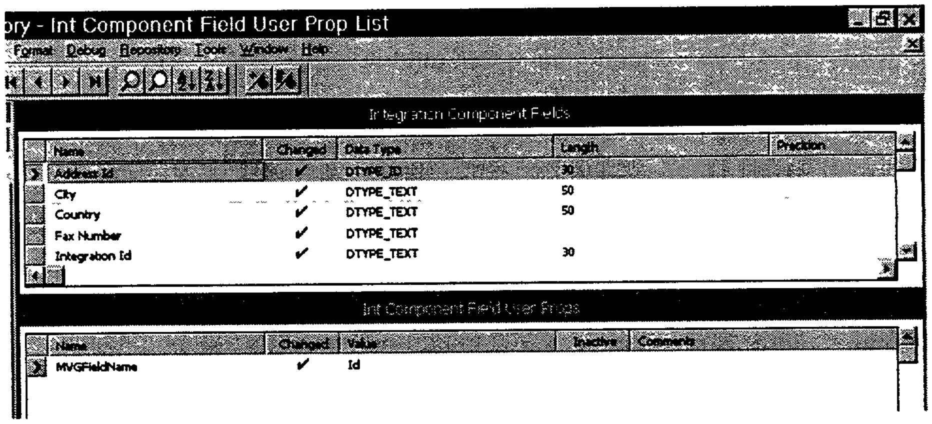
External Name Context	Name	Changed	Parent Integration Component	External Name
Account	Account	✓		Account
Account_Business Address	Account_Business Address	✓	Account	Business Address
Business Address	Business Address	✓	Account	Business Address

Name	Changed	Data Type	Length	Precision
Address Id	✓	DTYPE_ID	30	
City	✓	DTYPE_TEXT	50	
Country	✓	DTYPE_TEXT	50	
Fax Number	✓	DTYPE_TEXT		
Integration Id	✓	DTYPE_TEXT	30	
Phone Number	✓	DTYPE_TEXT		
Postal Code	✓	DTYPE_TEXT	30	
Province	✓	DTYPE_TEXT	50	
State	✓	DTYPE_TEXT	10	
Street Address	✓	DTYPE_TEXT	200	

Figure 2-16. Integration Component Fields Representing MVG

To represent both names, each field is assigned an integration component field user property that contains the entry *MVGFieldName*—or *AssocFieldName* if the user property is *MVGAssoc*. Its value is the name of the field shown in the parent business component—in this example, *Business Address*.

The value of *Id* in the Value field in Figure 2-17 is the name of the field in the parent business component, *Business Address*.


ory - Int Component Field User Prop List
Format Debug Repository Tools Window Help

Integration Component Fields

Name	Changed	Data Type	Length	Precision
Address Id	✓	DTYPE_ID	30	
City	✓	DTYPE_TEXT	50	
Country	✓	DTYPE_TEXT	50	
Fax Number	✓	DTYPE_TEXT		
Integration Id	✓	DTYPE_TEXT	30	

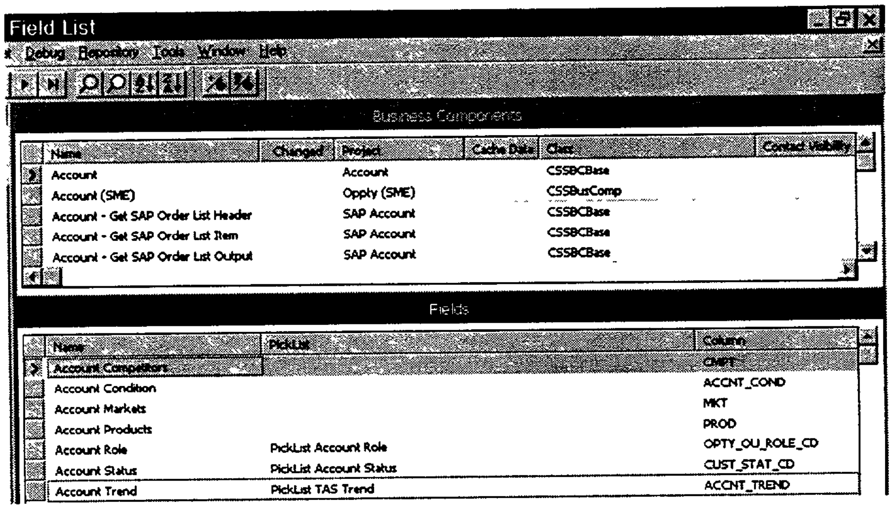
Int Component Field User Props

Name	Changed	Value	Inactive	Comments
MVGFieldName	✓	Id		

Figure 2-17. Integration Component Field User Property for Address Id Field

Representing Pick Lists

If an integration component field is created for a Siebel business field, and the business field is based on a pick list, as shown in the example in Figure 2-18, the integration component field must have a user property with the name of *PICKLIST* and a value of *Y*.

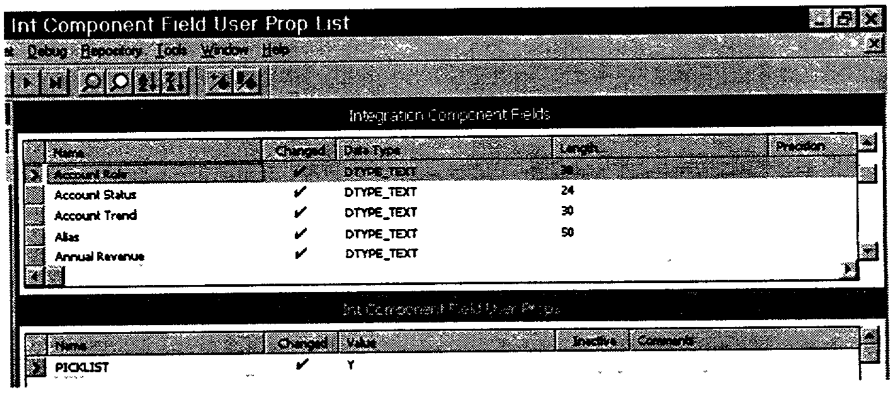


Name	Changed	Project	Cache Data	Class	Contact Visibility
Account		Account		CSSBCBase	
Account (SME)		Oppty (SME)		CSSBusComp	
Account - Get SAP Order List Header		SAP Account		CSSBCBase	
Account - Get SAP Order List Item		SAP Account		CSSBCBase	
Account - Get SAP Order List Output		SAP Account		CSSBCBase	

Name	PickList	Column
Account Competitors		CMPT
Account Condition		ACCNT_COND
Account Markets		MKT
Account Products		PROD
Account Role	PickList Account Role	OPTY_OU_ROLE_CD
Account Status	PickList Account Status	CUST_STAT_CD
Account Trend	PickList TAS Trend	ACCNT_TREND

Figure 2-18. Pick List Fields in Account Business Component

The corresponding integration component field must have a user property named *PICKLIST* with a value of *Y*. Figure 2-19 shows an example in Siebel Tools of the integration component field Account Role, with a user property *PICKLIST* set to a value of *Y*.


Int Component Field User Prop List
Debug Repository Tools Window Help
Integration Component Fields
Int Component Field User Props

Name	Changed	Data Type	Length	Precision
Account Role	✓	DTYPE_TEXT	30	
Account Status	✓	DTYPE_TEXT	24	
Account Trend	✓	DTYPE_TEXT	30	
Alias	✓	DTYPE_TEXT	50	
Annual Revenue	✓	DTYPE_TEXT		

Name	Changed	Value	Inactive	Comments
PICKLIST	✓	Y		

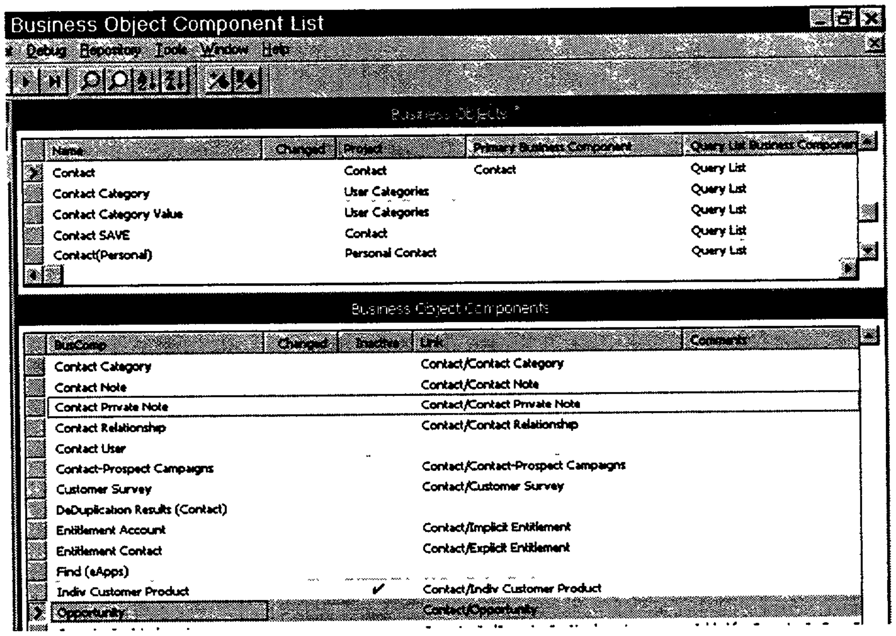
Figure 2-19. PICKLIST User Property on Integration Component Field Account Role

Representing Associations

Siebel business objects are made up of business components that are connected by a *link*. An *association* is a business component that represents the intersection table that contains these links.

The integration component definition of associations is similar to that of MVGs. User properties *Association* and *MVGAssociation* on the integration component denote that the corresponding business component is an associated business component or an associated MVG, respectively. For fields that are defined on MVG associations, *External Name* denotes the name of the business component field as it appears on the parent business component, and the user property *AssocFieldName* denotes the name of the business component field as it appears on the MVG business component.

In Figure 2-20, for example, the Contact business object comprises, at least partly, the Contact and Opportunity business components. The association between these two business components is represented by the Contact/Opportunity link.


Business Object Component List
Debug Repository Tools Window Help

Business Objects

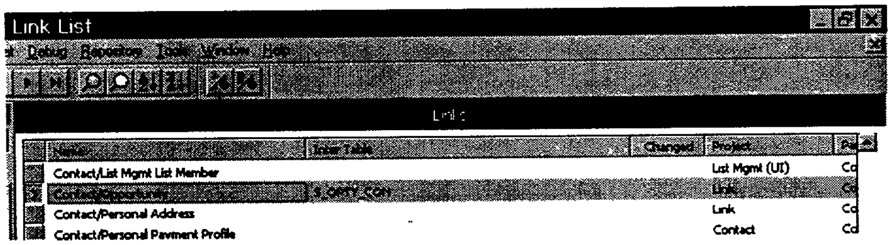
Name	Changed	Project	Primary Business Component	Query List Business Component
Contact		Contact	Contact	Query List
Contact Category		User Categories		Query List
Contact Category Value		User Categories		Query List
Contact SAVE		Contact		Query List
Contact(Personal)		Personal Contact		Query List

Business Object Components

BusComp	Changed	Inactive	Link	Comments
Contact Category			Contact/Contact Category	
Contact Note			Contact/Contact Note	
Contact Private Note			Contact/Contact Private Note	
Contact Relationship			Contact/Contact Relationship	
Contact User				
Contact-Prospect Campaigns			Contact/Contact-Prospect Campaigns	
Customer Survey			Contact/Customer Survey	
DeDuplication Results (Contact)				
Entitlement Account			Contact/Implicit Entitlement	
Entitlement Contact			Contact/Explicit Entitlement	
Find (eApps)				
Indiv Customer Product		✓	Contact/Indiv Customer Product	
Opportunity			Contact/Opportunity	

Figure 2-20. Contact and Opportunity Link in Contact Business Object

The link is defined in the Link List as having a value—that is, a table name—in the Intersection Table column, as shown in Figure 2-21.

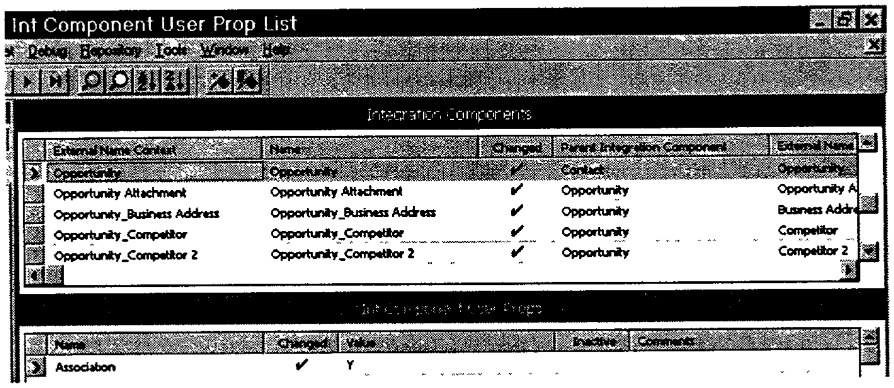


Links

Name	Inter Table	Changed	Project	Pa
Contact/List Mgmt List Member			List Mgmt (UI)	Co
Contact/Opportunity	S_OPTY_CON		Link	Co
Contact/Personal Address			Link	Co
Contact/Personal Payment Profile			Contact	Co

Figure 2-21. Non-Null Value in the Intersection Table Column for the Link

The Siebel Integration Object Builder creates a new integration component for the integration object—in this example, based on the Contact business object—that represents the association. This integration component—shown in Figure 2-22 as the Opportunity integration component—has one user property defined: the *Association* user property set to a value of *Y*.


Int Component User Prop List
Debug Repository Tools Window Help
Integration Components

External Name Context	Name	Changed	Parent Integration Component	External Name
Opportunity	Opportunity	✓	Contact	Opportunity
Opportunity Attachment	Opportunity Attachment	✓	Opportunity	Opportunity A
Opportunity_Business Address	Opportunity_Business Address	✓	Opportunity	Business Addr
Opportunity_Competitor	Opportunity_Competitor	✓	Opportunity	Competitor
Opportunity_Competitor 2	Opportunity_Competitor 2	✓	Opportunity	Competitor 2

Name	Changed	Value	Inactive	Comments
Association	✓	Y		

Figure 2-22. Integration Component Representation of Association

Understanding User Keys

User keys uniquely identify Siebel business components. In the context of Siebel business objects, user keys are a group of fields whose values must uniquely identify only one Siebel business component record. Integration components within a corresponding integration object also contain user keys.

User keys play a valuable role in the operation of an integration component. For example, if a Siebel business component has new values to set in the database, user keys can be used to determine whether or not to perform an insert or update.

Integration component user keys are built by the Siebel Integration Object Builder based on values in the underlying table of the business component upon which the integration component is based.

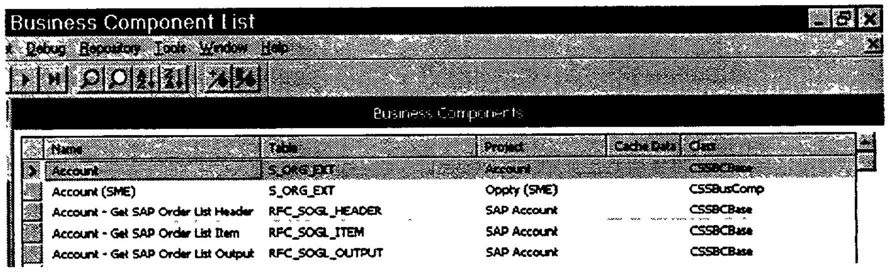
NOTE: Integration objects that represent Siebel business objects, and that are to be used in Insert or Update, Synchronize, or Execute operations, must have at least one integration component user key defined.

In Siebel Tools, the Integration Component User Key specifies which of the integration component fields should be used for a user key.

The Siebel Integration Object Builder computes the user keys by traversing several other Siebel types, including the business object, business component, table, and link types. Not all user keys, however, will meet the requirements to be used as the basis for integration object user keys.

For the sake of understanding how the Siebel Integration Object Builder determines valid integration component user keys, you can simulate the process of validating the user keys.

For example, determine the table on which the your business component is based. In Siebel Tools, you can look up this information yourself. Navigate to the Business Components screen and select a business component. For example, Figure 2-23 shows that S_ORG_EXT is the table on which the Account business component is based.

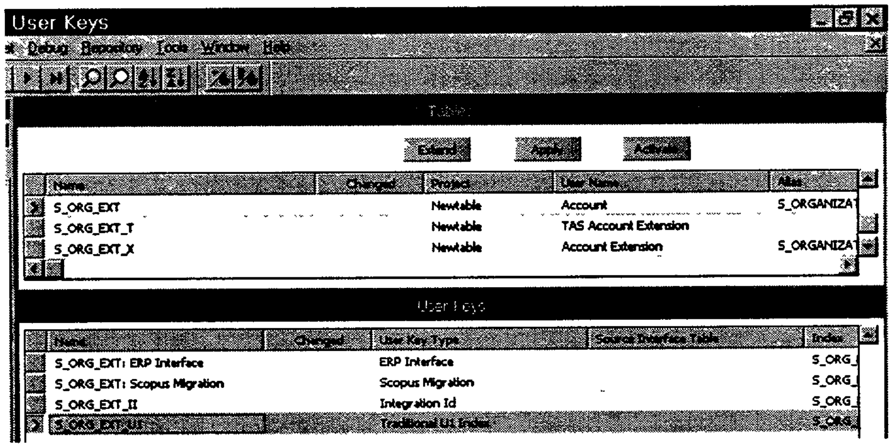


Name	Table	Project	Cache Data	Class
Account	S_ORG_EXT	Account		CSSBCBase
Account (SME)	S_ORG_EXT	Oppty (SME)		CSSBusComp
Account - Get SAP Order List Header	RFC_SOGL_HEADER	SAP Account		CSSBCBase
Account - Get SAP Order List Item	RFC_SOGL_ITEM	SAP Account		CSSBCBase
Account - Get SAP Order List Output	RFC_SOGL_OUTPUT	SAP Account		CSSBCBase

Figure 2-23. Table S_ORG_EXT and Account Business Component

You can then navigate to the Tables screen, locate the table you want—in this example, S_ORG_EXT—and open the User Keys applet to see the user keys defined for that table.

For example, as shown in Figure 2-24, the table S_ORG_EXT has four user keys.



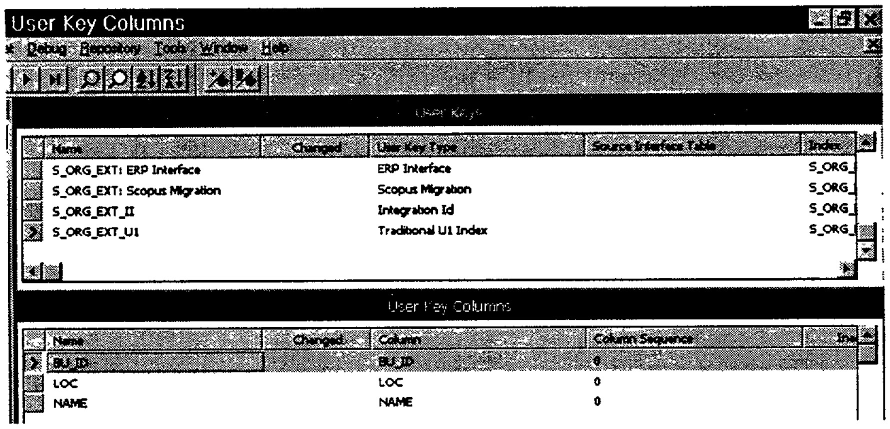
Name	Changed	Project	User Name	Alias
S_ORG_EXT		Newtable	Account	S_ORGANIZA
S_ORG_EXT_T		Newtable	TAS Account Extension	
S_ORG_EXT_X		Newtable	Account Extension	S_ORGANIZA

Name	Changed	User Key Type	Source Interface Table	Index
S_ORG_EXT: ERP Interface		ERP Interface		S_ORG_
S_ORG_EXT: Scopus Migration		Scopus Migration		S_ORG_
S_ORG_EXT_II		Integration Id		S_ORG_
S_ORG_EXT_U1		Traditional U1 Index		S_ORG_

Figure 2-24. User Keys for Table S_ORG_EXT

Not all user keys will necessarily be valid for a given business component. Multiple business components can map to the same underlying table; therefore, it is possible that a table's user key is not valid for a particular business component but is specific to another business component.

Each User Key Column defined for a given user key must be exposed to the business component you are interested in. For example, Figure 2-25 shows three user key columns for the user key S_ORG_EXT_UI.

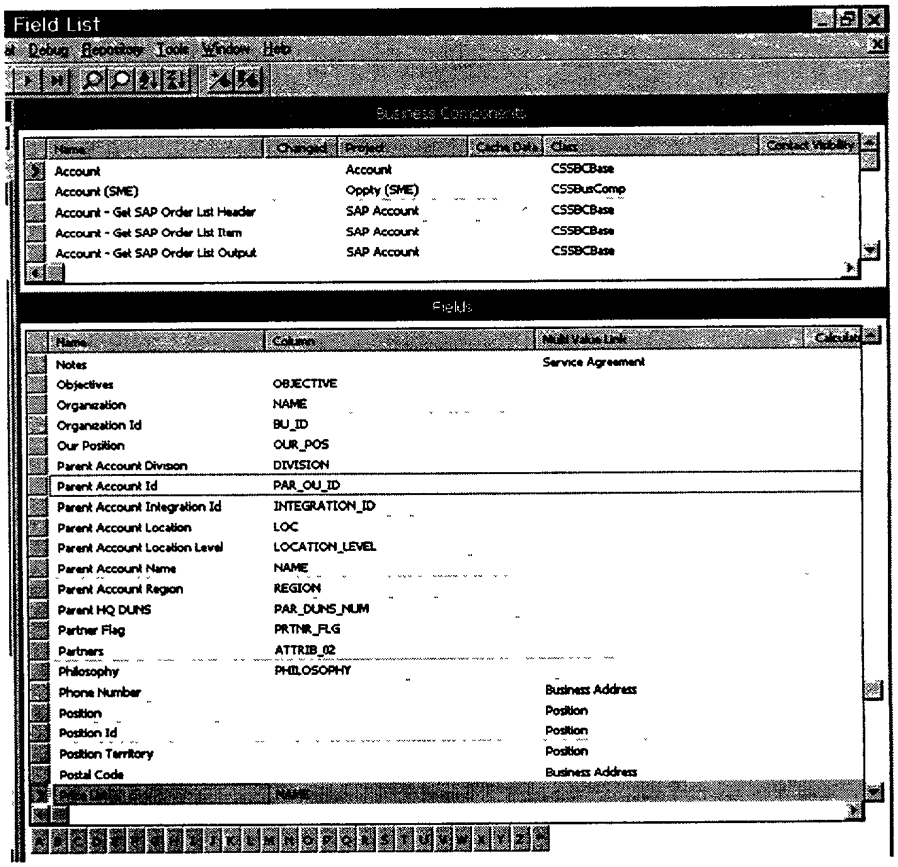


Name	Changed	User Key Type	Source Interface Table	Index
S_ORG_EXT: ERP Interface		ERP Interface		S_ORG_
S_ORG_EXT: Scopus Migration		Scopus Migration		S_ORG_
S_ORG_EXT_II		Integration Id		S_ORG_
S_ORG_EXT_U1		Traditional U1 Index		S_ORG_

Name	Changed	Column	Column Sequence	Ina
BU_ID		BU_ID	0	
LOC		LOC	0	
NAME		NAME	0	

Figure 2-25. User Key Columns for the S_ORG_EXT_UI User Key

If all of the user key columns are exposed in the business component and those columns are not foreign keys, the Siebel Integration Object Builder creates an integration component user key based on the table's user key. The Siebel Integration Object Builder also defines one integration component user key field corresponding to each of the table's user key columns. For example, in Figure 2-26, the three user key columns are exposed in the Integration Component Fields applet for the Account integration component.



Business Components

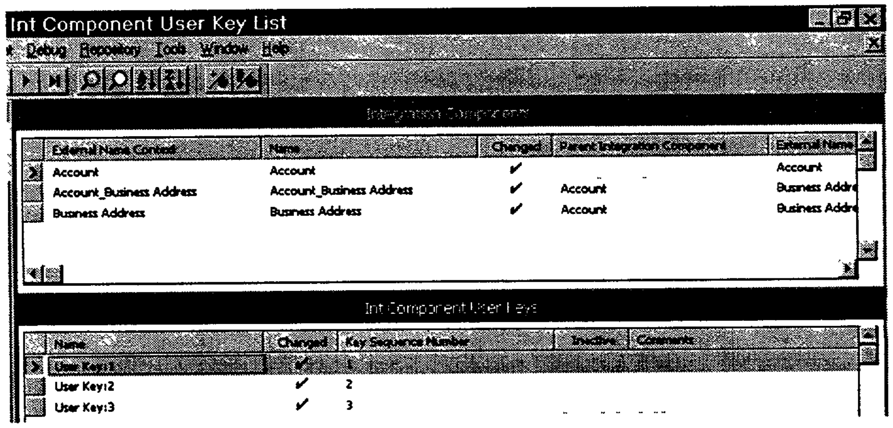
Name	Changed	Project	Cache Data	Class	Contact Visibility
Account		Account		CSSBCBase	
Account (SME)		Oppty (SME)		CSSBusComp	
Account - Get SAP Order List Header		SAP Account		CSSBCBase	
Account - Get SAP Order List Item		SAP Account		CSSBCBase	
Account - Get SAP Order List Output		SAP Account		CSSBCBase	

Fields

Name	Column	Multi Value Link	Calculat
Notes		Service Agreement	
Objectives	OBJECTIVE		
Organization	NAME		
Organization Id	BU_ID		
Our Position	OUR_POS		
Parent Account Divison	DIVISION		
Parent Account Id	PAR_OU_ID		
Parent Account Integration Id	INTEGRATION_ID		
Parent Account Location	LOC		
Parent Account Location Level	LOCATION_LEVEL		
Parent Account Name	NAME		
Parent Account Region	REGION		
Parent HQ DUNS	PAR_DUNS_NUM		
Partner Flag	PRTNR_FLG		
Partners	ATTRIB_02		
Philosophy	PHILOSOPHY		
Phone Number		Business Address	
Position		Position	
Position Id		Position	
Position Territory		Position	
Postal Code		Business Address	
[illegible]	NAME		

Figure 2-26. Integration Component Field List Containing User Key Columns

The user key column BU_ID is a foreign key. User key columns LOC and NAME are exposed in the integration component Fields list and *are not* foreign keys. The Siebel Integration Object Builder, in this case, builds the integration component user keys based on these table user keys. It defines one integration component user key field for each table user key column, as illustrated in Figure 2-27.

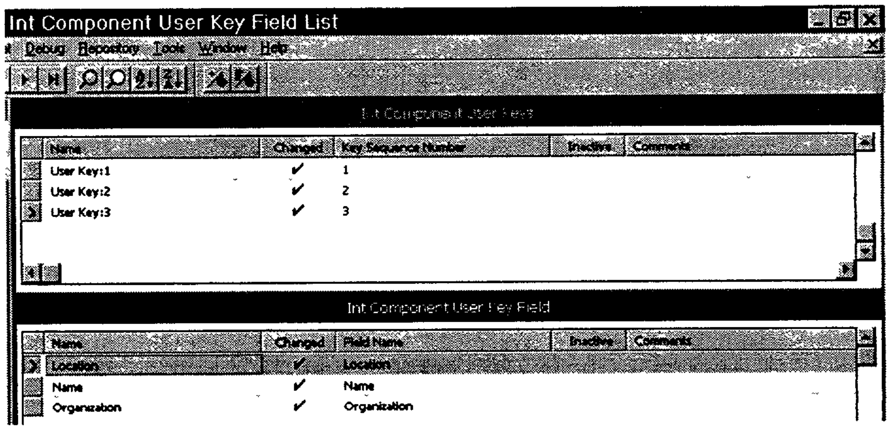

Int Component User Key List
Debug Repository Tools Window Help
Integration Components
Int Component User Keys

External Name Context	Name	Changed	Parent Integration Component	External Name
Account	Account	✓		Account
Account_Business Address	Account_Business Address	✓	Account	Business Addre
Business Address	Business Address	✓	Account	Business Addre

Name	Changed	Key Sequence Number	Inactive	Comments
User Key:1	✓	1		
User Key:2	✓	2		
User Key:3	✓	3		

Figure 2-27. Integration Component User Keys for Each Table User Key Column

Each valid integration component user key will contain fields. For example, as shown in Figure 2-28, User Key 3, for the Account integration component, is made up of three fields: Location, Name, and Organization.


Int Component User Key Field List
Debug Repository Tools Window Help
Int Component User Keys
Int Component User Key Field

Name	Changed	Key Sequence Number	Inactive	Comments
User Key:1	✓	1		
User Key:2	✓	2		
User Key:3	✓	3		

Name	Changed	Field Name	Inactive	Comments
Location	✓	Location		
Name	✓	Name		
Organization	✓	Organization		

Figure 2-28. Account Integration Component User Key Fields

When the Siebel Integration Object Builder creates these integration component user keys, it makes every attempt to use the appropriate table user keys; that is, the user keys that will help uniquely identify a given record. In some cases, you might find that certain integration component user keys created by the Siebel Integration Object Builder are not useful for your particular needs. In that case, you can manually inactivate the keys you do not want to use by checking the Inactive flag on that particular user key in Siebel Tools. You can also inactivate user key fields within a given user key.

NOTE: It is important to remember that user keys are groups of fields that, in combination, uniquely identify business components. By inactivating integration component user keys or user key fields, you might cause unexpected and undesirable results.

Setting User Properties on Integration Objects

You can define user properties for your integration objects. These user properties help determine special processing and behavioral requirements of integration objects for a specific EAI adapter.

The Level column shown in Table 2-2 and Table 2-3 on page 2-44 can take the following values:

- *O*, for Object Level
- *C*, for Component Level
- *F*, for Field Level

Table 2-2. User Properties for Integration Objects Based on Siebel Business Objects (1 of 2)

User Property	Allowable Values	Level	Description
PICKLIST	Y	F	Integration field is based on a pick list.
Association	Y	C	Integration component is an association business component.
MVG	Y	C	Integration component is an MVG business component.
MVGAssociation	Y	C	Integration component is an association MVG business component.
MVGFieldName	Valid field name	F	If the component that owns this integration field is based on an MVG, the value of this user property gives the name of the business component field as the MVG component knows it.
AssocFieldName	Valid field name	F	If the component that owns this integration field is based on an MVGAssociation, the value of this user property gives the name of the business component field as the Association MVG component knows it.

Table 2-2. User Properties for Integration Objects Based on Siebel Business Objects (2 of 2)

User Property	Allowable Values	Level	Description
Ignore Bounded Picklist	Y	OCF	Ignores errors in which a value is given in an integration object instance for an integration field based on a bounded pick list, where the value given does not match a value in the pick list. The default behavior is to report an error. If specified at the Object level, all fields in all components that are based on bounded pick lists will ignore the error.
NoInsert, NoDelete	Y	C	Instructs the EAI Siebel Adapter that it should not perform inserts or deletes, respectively, on the business component that the integration component represents.
NoUpdate	Y	CF	Instructs the EAI Siebel Adapter that it should not perform an update on the business component or field that the integration object or field represents. If NoUpdate is set at the Component level, no fields in that component can be updated.
FieldDependencyXXX	Valid integration field	F	Defines a dependency between the integration field that has this user property and the integration field specified by the value of the user property. Multiple dependencies are specified by separate user property entries. The field specified in the value must be from the same component as the field that has the user property.
AdminMode	Y	C	Sets Admin mode on the business component. For more information, see the *Siebel Object Types Reference*.

Table 2-3 shows the user properties for integration objects that represent SAP IDOCs.

Table 2-3. User Properties for Integration Objects Based on SAP IDOCs

User Property	Allowable Values	Level	Description
Logical Message Type	Logical message type corresponding to the IDOC	O	This user property is ignored by the SAP adapter in the inbound direction—receiving IDOCs from SAP. In the outbound direction—sending IDOCs to SAP—the SAP adapter populates the Control segment MESTYP field using the value of this user property before sending each IDOC to SAP.
Offset	integer	F	This is the character offset for the IDOC field within its corresponding IDOC segment. The SAP adapter looks at this user property in both inbound and outbound directions. In the inbound direction, the SAP adapter looks at this offset in the raw IDOC buffer to populate the integration field. This is IDOC parsing. In the outbound direction, the SAP outbound adapter inserts the value corresponding to the integration field into the raw IDOC buffer at this offset.

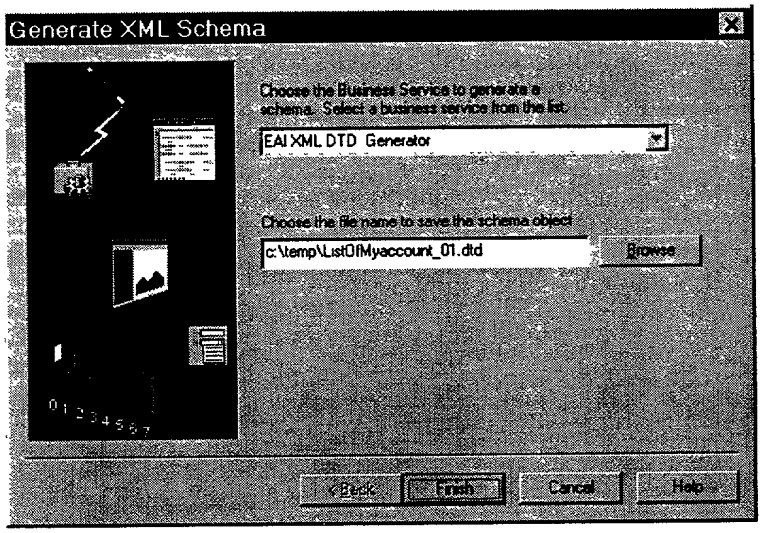
Generating Schema

Generating the schema of an integration object is another task you might want to perform at certain points in your integration project. You perform this task in the same way for any type of integration object. The difference in integration objects will determine what you do with the output schema.

To generate an integration object schema

1. In Siebel Tools, click on an integration object to make it the active object.
2. Click Generate Schema.

   This displays the first page of the Generate Schema wizard.


Generate XML Schema

Choose the Business Service to generate a schema. Select a business service from the list.

EAI XML DTD Generator

Choose the file name to save the schema object

c:\temp\ListOfMyaccount_01.dtd

3. Choose the EAI XML DTD Generator business service.
4. Choose a location where you want to save the resulting file.
5. Click Finish.

   The wizard generates an XML DTD of the integration object you selected. You can use this DTD to help you map external data directly to the integration object. The DTD serves as the definition for the XML elements you can create using an external application or XML editing tool.

Using the EAI Siebel Adapter

Exhibit D

Using the EAI Siebel Adapter

The EAI Siebel Adapter is a general-purpose integration business service that allows you to:

- Read a Siebel business object from the Siebel data model into an integration object.
- Write an integration object whose data originates externally into a Siebel business object.

The EAI Siebel Adapter is based on the *CSSEAISiebelAdapterService* class.

Using the EAI Siebel Adapter Methods

The EAI Siebel Adapter supports the following methods:

- Upsert
- Query
- Delete
- Synchronize
- Execute

Understanding EAI Siebel Adapter Method Arguments

Each of the EAI Siebel Adapter methods takes arguments that allow you to specify required and optional information to the adapter. The EAI Siebel Adapter methods share some of the same arguments. You can locate the arguments for each method in Table 4-1.

Table 4-1. EAI Siebel Adapter Method Arguments

Argument	Upsert	Execute	Query	Delete	Synchronize
PrimaryRowId	■	■	■	■	■
SiebelMessage	■	■	■	■	■
OutputIntObjectFormat		■	■		
OutputIntObjectName		■	■		
IntObjectName				■	

Table 4-2 describes each argument of the EAI Siebel Adapter methods.

Table 4-2. EAI Siebel Adapter Method Arguments

Argument	Type	Description
IntObjectName	Input	The name of the integration object that is to be deleted.
NumOutputObjects	Output	Number of output integration objects.
OutputIntObjectFormat	Input	The data format for output of the integration object that represents a Siebel business object. By default the format is Siebel Hierarchical. You can set this to one of the following formats: ■ Siebel Hierarchical ■ XML Hierarchical ■ XML Flat
OutputIntObjectName	Input	The name of the integration object that is to be output.
PrimaryRowId	Input/Output	The primary row ID that corresponds to either the Siebel business component field row ID that is the primary key for your primary business component, or the row ID of whatever column or field serves as a primary key or identifier in your external data source.
SiebelMessage	Input/Output	A child property set.

Upsert Method

The Upsert method is a combination of update and insert methods. The Upsert method performs one of the following operations against a business object instance based on the results of a query against the business object instance.

- If the initial Upsert query against the integration component fails, the Upsert method inserts the child property set indicated by the SiebelMessage argument into the Siebel database.
- If the child integration component exists and the PrimaryRowId field value matches the primary row ID of the data source, the business component row corresponding to the PrimaryRowId is updated with the values in the integration component instance.

Query Method

You pass the Query method an integration object instance or a Primary Row Id. The adapter uses the integration object or the PrimaryRowId—depending on which was passed to the method—as criteria to query the base business objects and to return a corresponding integration object instance.

Delete Method

You can delete a business component instance, given an integration object. A business component is deleted as specified by an integration object. The fields of an integration component instance are used to query a business component. If the property BatchMode is set to `true` on the root property set, the user key fields in the integration component instance are used to select a business component row and perform a subsequent delete.

NOTE: If you want the EAI Siebel Adapter to perform a delete operation, you should define an integration object that contains the minimum fields on the primary business component for the business object.

Synchronize Method

You can use the Synchronize method to make the values in a business object instance match those of an integration object instance. This operation can result in updates, inserts, or deletes on business components. Some rules apply to the results of this method:

- If all child components of a type are missing from an integration object, the corresponding business component rows are left untouched.
- If some child components of a particular type are present, the business component rows corresponding to them are updated/created, and any business component row that does not have a corresponding integration component instance is deleted.

Execute Method

The Execute method can be specified on the Siebel Adapter to perform combinations of various operations of components in an integration object instance. This method will be used to execute on the following operations:

- Delete
- Upsert
- Synchronize
- Query

An XML document that is sent to Siebel eBusiness Application can include "operations" that describe whether a particular data element needs to be inserted, updated, deleted, or synchronized. These operations can be specified as an attribute at the component level. They cannot be specified for any other element.

To Specify an Operation

Specify an attribute named operation, all lowercase, to the component's XML element. The legal values for this attribute are:

- upsert
- sync
- delete
- query

NOTE: Specifying operations at the component level, ListOf tag, is not supported.

Delete a Child or a Grandchild Component

Specify the Execute method on the Siebel Adapter and specify a delete operation on that component and the upsert operation on all its parents.

Following is an example demonstrating an operation of upsert and delete:

```
<SiebelMessage MessageId="" MessageType="Integration Object"
IntObjectName="Sample Account">

   <ListofSampleAccount>

      <Account operation="upsert">

         <Name>A. K. Parker Distribution</Name>

         <Location>HQ-Distribution</Location>

        <Organization>North American Organization</Organization>

         <Division/>

         <CurrencyCode>USD</CurrencyCode>

         <Description>This is the key account in the AK Parker
             Family</Description>

         <HomePage>www.parker.com</HomePage>

         <LineofBusiness>Manufacturing</LineofBusiness>
```

```
                <ListOfContact>
                   <Contact operation="delete">
                      <FirstName>Stan</FirstName>
                      <JobTitle>Senior Mgr of MIS</JobTitle>
                      <LastName>Graner</LastName>
                      <MiddleName>A</MiddleName>
                      <PersonalContact>N</PersonalContact>
                      <Account>A. K. Parker Distribution</Account>
                     <AccountLocation>HQ-Distribution</AccountLocation>
                   </Contact>
                </ListOfContact>
             </Account>
          </ListofSampleAccount>
       </SiebelMessage>
```

Table 4-3 lists the supported operations for the root component as well as child components.

Table 4-3. Supported Operations

Root Component Operation	Child Component Operations
upsert	delete, upsert
delete	No operation should be specified.
sync	No operation should be specified. Specifying a sync at the component level would work as specifying upsert at the child component level.
query	query

Representing Multi-Value Groups in the EAI Siebel Adapter

Multi-value groups (MVGs) are mapped to integration component instances. MVGs can be specified by creating a user property called MVG on the integration component and setting it to Y.

Also, an integration component instance that corresponds to a primary MVG is denoted by a field *IsPrimaryMVG* set to Y. This field does not have a corresponding integration component field in the metadata.

Each MVG that appears on the client UI is mapped to an integration component. For example, in the Orders - Sales Orders - Terms screen, you will see an account address, a bill-to address, and a ship-to address. Each MVG maps to a separate integration component definition.

Representing Pick List Values

Fields that correspond to a pick list map to integration component fields contained in the integration component. These integration component represent the business component containing the pick list. A user property call PICKLIST is set to Y on the integration component field that corresponds to a field in a pick list business component.

Service Arguments Tracing

The Siebel Adapter takes a Property Set as an input and outputs another Property Set. We recommend saving the input and output of the Siebel Adapter for debugging purposes. A new Logging Event, EAI Service Argument Tracing, should be used to accomplish this task, as described below:

Level 3 - will cause the Input Property Set to be converted to XML and written to a file in the log directory in case of an error in the Siebel Adapter.

Level 4 - will write the Input and Output Property Sets to files in the log directory every time Siebel Adapter is called.

There is also an entry made in the server component or the client log file, depending on where the Siebel Adapter was called, that specifies the filename(s) where arguments were written. The entry will be a DumpFile event, all on one line, as shown below:

```
DumpFile        DumpFileOpen    3       2000-09-12 23:45:27
e:\v70_10513\log\EAISiebAdpt_input_args_20522239(002).dmp:'EAI
Siebel Adapter' Business Service input arguments
```

The above entry means that Siebel Adapter input arguments were written to file.

```
e:\v70_10513\log\EAISiebAdpt_input_args_20522239(002).dmp
```

This file can be used as an input to Siebel Adapter in Business Service Simulator the to reproduce the error.

For information on setting logging events, please see the *Siebel Server Administration Guide.*